Architectural Knowledge Management: Decision Guidance in Service-Oriented Architecture Design

Dr. Olaf Zimmermann
Executive IT Architect

olz@zurich.ibm.com
The SEI Architecture Technology User Network (SATURN) Conference brings together experts to exchange best architecture-centric practices in developing, acquiring, and maintaining software-reliant systems.
SEI Launches New Technology Blog

With posts written by staff members, the blog will provide the SEI audience with insights into the broad spectrum of work at the SEI via a two-way, read-write medium.

http://blog.sei.cmu.edu/
How to Participate Today

Open and close your Panel
View, Select, and Test your audio
Submit text questions
Q&A addressed at the end of today’s session
Today’s Speaker

Dr. Olaf Zimmermann is the leader for the Architectural Knowledge Management theme at the SEI Architecture Technology User Network (SATURN 2011) Conference. He is also a research staff member at IBM Research in Zurich, Switzerland. His research interests include architectural knowledge management and service-oriented architecture design. For his doctoral dissertation work at Stuttgart University in 2009, he created an architectural decision modeling framework for service-oriented architecture design. From 1999 to 2005, Zimmermann worked as a solution architect, helping IBM clients designing SOA/web services and Java Enterprise Edition (JEE) solutions on professional services projects. He also educated practitioners around the world on emerging middleware technologies. In the beginning of his career, Zimmermann worked as a scientific consultant in the IBM European Networking Center (ENC) in Heidelberg, Germany, focusing on industry-specific middleware frameworks for systems and network management. He is a regular conference speaker and an author of the Springer text book Perspectives on Web Services. He contributed to several IBM Redbooks including the first Redbook on Eclipse and Web services authored in 2001. Olaf received a graduate "Diplom-Informatiker" degree in computer science from the Technical University in Braunschweig, Germany, in 1993. He is an Open Group Distinguished Certified IT Architect and IBM Senior Certified IT Architect.
Agenda

- SOA principles and patterns
- Case studies
- Architectural decisions
- SOA Decision Modeling (SOAD) assets
- Discussion and summary
Agenda

- SOA principles and patterns
- Case studies
- Architectural decisions
- SOA Decision Modeling (SOAD) assets
- Discussion and summary
What is a Service-Oriented Architecture (SOA)?

No single definition – “SOA is different things to different people”

- A set of services that a business wants to expose to their customers and partners, or other portions of the organization.

- An architectural style which requires a service provider (a.k.a. server) and a service requestor (a.k.a. consumer or client).

- A set of architectural patterns such as service consumer-provider contract, enterprise service bus, service composition, and service registry, promoting principles such as modularity, layering, and loose coupling to achieve design goals such as separation of concerns, reuse, and flexibility.

- A programming and deployment model realized by standards, tools and technologies such as Web services and Service Component Architecture (SCA).
Agenda

- SOA principles and patterns
- Case studies
- Architectural decisions
- SOA Decision Modeling (SOAD) assets
- Discussion and summary
Case 1: Core Banking SOA with Web Services

IBM WebSphere® (pSeries)

Java Client
.NET Client
Browser
Office

Web Services Adapter Layer

Java™ API

Java Backend Connectors (IBM WebSphere MQ, CICS®)

Access Layer

Business Function

Database
(IBM DB2®)

Web Application

SOAP

Repository

generate

Documentation

generate

WSDL

IBM CICS
(zSeries)

Platform independent
Case 2: Multi-Channel Order Management (B2B)

- **Functional domain**
  - Order entry management
  - Two business processes: new customer, relocation
  - Main SOA drivers: deeper automation grade, share services between domains

- **Service design**
  - Top-down from requirement and bottom-up from existing wholesaler systems
  - Recurring architectural decisions:
    - Protocol choices
    - Transactionality
    - Security policies
    - Interface granularity
Agenda

- SOA principles and patterns
- Case studies
- **Architectural decisions**
- SOA Decision Modeling (SOAD) assets
- Discussion and summary
What are Architectural Decisions? Why Bother?

- “The design decisions that are costly to change” (Grady Booch, 2009)

  
  “Architectural decisions capture key design issues and the rationale behind chosen solutions. They are conscious design decisions concerning a software-intensive system as a whole or one or more of its core components and connectors in any given view. The outcome of architectural decisions influences the system’s nonfunctional characteristics including its software quality attributes.”

- From IBM UMF work product description ART 0513 (previous name: ARC 100):
  
  “The purpose of the Architectural Decisions work product is to:
  – Provide a single place to find important architectural decisions
  – Make explicit the rationale and justification of architectural decisions
  – Preserve design integrity in the provision of functionality and its allocation to system components
  – Ensure that the architecture is extensible and can support an evolving system
  – Provide a reference of documented decisions for new people who join the project
  – Avoid unnecessary reconsideration of the same issues”
## Architectural Decision (AD) about Integration Style – Documented according to IBM Unified Method Framework (UMF)

<table>
<thead>
<tr>
<th>Subject Area</th>
<th>Process and service layer design</th>
<th>Topic</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Integration Style</td>
<td>AD ID</td>
<td>3</td>
</tr>
<tr>
<td>Decision Made</td>
<td>We decided for RPC and the Messaging pattern (Enterprise Integration Patterns)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue or Problem</td>
<td>How should process activities and underlying services communicate?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assumptions</td>
<td>Process model and requirements NFR 1 to NFR 7 are valid and stable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motivation</td>
<td>If logical layers are physically distributed, they must be integrated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternatives</td>
<td>File transfer, shared database, no physical distribution (local calls)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Justification</td>
<td>This is an inherently synchronous scenario: VSP users as well as internal T staff expect immediate responses to their requests (NFR 5). Messaging will give us guaranteed delivery (NFR 3, NFR 6).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implications</td>
<td>Need to select, install, and configure a message-oriented middleware.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived Requirements</td>
<td>Many finer grained patterns are now eligible and have to be decided upon: message construction, channel design, message routing, message transformation, system management (see Enterprise Integration Patterns book).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related Decisions</td>
<td>Next, we have to decide on one or more integration technologies implementing the selected two integration styles. Many alternatives exist, e.g., Java Message Service (JMS) providers.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

- SOA principles and patterns
- Case studies
- Architectural decisions
- SOA Decision Modeling (SOAD) assets
- Discussion and summary
“Model View Controller (MVC) is a common architectural pattern to control the Web page flow.”

“We decided for the MVC alternative to resolve the web page flow issue because we gained positive experience with it on many similar projects.”
From AD Documentation to Active Method Guidance

600+ Decisions in SOA Decision Guidance Model (SOAD)

- **Business Executive Level**
  - Architectural Style (SOA or other?)

- **Conceptual Level**
  - Service Composition Paradigm (Processes? Classes?)
  - Process-Enabled SOA
  - Message Exchange Pattern (Request-Reply? One Way?)

- **Technology Level**
  - Workflow Language (BPEL? Other?)
  - Transport Protocol (SOAP over HTTP?)
  - BPEL 2.0
  - SOAP/HTTP 1.1

- **Vendor Asset Level**
  - BPEL Engine (IBM WPS? Other?)
  - SOAP Engine (Apache Axis2?)
  - IBM WebSphere Transaction Settings?
  - Eclipse Web Tools Project (WTP) Usage?
  - Apache/WebSphere Configuration?

- Architectural Decision Issue (with Alternatives)
  - Transaction Boundaries?
  - Service Granularity?
  - Message Confidentiality?
  - Transaction Qualifiers in SCA?
  - Operations per WSDL Port Type?
  - HTTPS or WSSE?

- Decision Made/Alternative Selected

For each project:
- Architectural Style (SOA or other?)

For each service:
- Process-Enabled SOA
- Synchronous Request-Reply

For each process:
- Workflow Language (BPEL? Other?)
- Transport Protocol (SOAP over HTTP?)
- BPEL 2.0
- SOAP/HTTP 1.1

- BPEL Engine (IBM WPS? Other?)
- SOAP Engine (Apache Axis2?)
AD Issue #1 – Addressing Service Granularity Topic

Decision drivers: Functional requirements (domain model), capabilities of BPEL, SOAP, WSDL, XML processors (verbosity), interoperability, network topology, number of deployment artifacts and generated code structure, strong vs. weak typing philosophy.

Issue: In Message Granularity (Conceptual/Technology Level)
How many message parts should be defined in the service contract?
How deep should the part elements be structured?
The four alternatives have not been published as patterns yet.

Alternative 1: Dot Pattern
Single scalar parameter
Easy to process for SOAP/XML engines, much work for programmer

Alternative 2: Bar Pattern
Single complex parameter
Deep structure and exotic types can cause interoperability issues.

Alternative 3: Dotted Line Pattern
Multiple scalar parameters
Handled by all common engines, some programmer convenience.

Alternative 4: Comb Pattern
Multiple complex parameters
Combination of options 2 and 3, biggest overhead for processing engines.

Recommendation: All alternatives have their place; alternatives 2 and 3 are often chosen. Base decision on layer and service type. Avoid overly deep nesting of data structures unless you want to stress test the XML processing. Minimize message verbosity.
Architectural Decisions Knowledge Tools

- Regulatory compliance
  - E.g., maturity models
- Collaboration
  - In geographically distributed teams
- Reuse
  - Of already gained knowledge
- Other required features:
  - Import and export
  - Searching and filtering
  - Dependency management
  - Report generation
Three Usage Scenarios for Architectural Decisions

**Scenario 1: After-the-Fact Decision Capturing**

1.1 Document decisions made and their rationale (supporting decision log report generation)

2.1 Distinguish decisions required (issues) from decisions made (outcomes)

2.2 Share issues and related best practices via guidance models

2.3 Assign design work items (issues with open outcomes) to team members and track decision making progress (a.k.a. “backlog”)

**Scenario 2: Active Method Guidance**

3.1 Identify issues in requirements artifacts and trace their resolution

3.2 Bind architectural decisions to enterprise-wide architectural principles and reusable assets such as pattern repositories

3.3 Enforce decisions in UML and topology models

3.4 Integrate with process tasks

3.5 Measure decision making practices (model analysis)

**Scenario 3: Cross Role Collaboration**

- Knowledge Engineer in Enterprise Architecture (EA) Group
- Solution Architect
- EA Group, Center of Excellence
- Project Team
- External Parties (e.g. Auditors)
Agenda

- SOA principles and patterns
- Case studies
- Architectural decisions
- SOA Decision Modeling (SOAD) assets
- Discussion and summary
SOA Lessons Learned

- SOA concepts and Web services technologies work
  - Interoperability proven
  - Performance not an issue

- Same old architecture?
  - Broker vs. Enterprise Service Bus
  - Workflow vs. service composition
  - Directory service vs. registry

- Not all patterns always have to be used
  - Judge and decide on a pattern-by-pattern base
  - Decision making driven by (non-)functional project/program requirements
  - Don’t confuse concepts and technology (SOA vs. Web services, REST)

- Follow a crawl, walk, run approach
  - As for any other non-trivial problem/solution
  - Don’t over-engineer (e.g. complex XML schema)
Summary and Discussion

- Architectural decision making is a key responsibility of IT architects which is often underestimated and underrepresented in existing methods and tools.

- In SOA design, many decisions recur. This makes it possible to reduce the documentation effort and to share architectural decision knowledge including best practices (design acceleration and quality assurance).

- Prototypical tool support for decision modeling with reuse is available.

- We would like to hear from you now…
  - … are the presented scenarios, concepts, and tool features useful and usable?
  - … would you have additional requirements, e.g. collaboration and integration needs?
Value of Architectural Decision Modeling

1. Improve standard compliance and audit readiness
   - Use standard template and tool to preserve design rationale
   - Decision model analytics (status)

2. Accelerate design work and reduce project risk
   - Active Method Guidance
     - Harvest design issues and reuse chosen solutions from previous projects
   - Compare made decisions with best practices recommendations
   - Guidance model analytics (usage)

3. Trace and govern design evolution from project initiation to go-live
   - After-the-Fact Decision Capturing
   - Cross Role Collaboration
     - Share enterprise-wide guidance models as assets
   - Link issues to requirements
   - Bind outcomes to UML and Zephyr model elements
   - Process enactment and health reports

Legend:
- Goal
- Scenario
- Practice
- Measure
More Information and Upcoming Events

- **SOAD overview article:**

- **SOAD tutorial and case study reports:**
  - [http://soadecisions.org/soad.htm](http://soadecisions.org/soad.htm)

- **Upcoming conferences with focus on Architectural Knowledge Management (AKM) and/or SOA/Web services design:**
  - SEI SATURN 2011 (May 16-20, Burlingame, California, USA)
    - Registration now open
  - IEEE WICSA 2011 (June 20-24, Boulder, Colorado, USA)
    - Paper submissions still accepted until Feb 14, 2011
  - IEEE ECOWS 2011 (Sept 14-16, Lugano, Switzerland)
    - Call for papers out and submissions accepted until April 15/April 30, 2011
Decision Guidance in Service-Oriented Architecture Design – Reference Material

Dr. Olaf Zimmermann
Executive IT Architect

olz@zurich.ibm.com
SOA Patterns Overview

- No industry consensus on SOA principles and patterns yet:
- Each author defines his/her own – many terminology mismatches

Decision Identification ➔ Decision Making ➔ Decision Enforcement

**Issue: Reference Architecture Selection (Business Executive Level)**

Which reference model should be used to define architectural layers?

Background reading: Krafzig et al. “Enterprise SOA”, Josuttis “SOA in Practice”

**Alternative 1:** SOMA/SOA Solution Model (IBM)
- Pros: Platform-neutral, comprehensive, widely adopted
- Cons: Rather abstract, has to be refined for projects and solutions

**Alternative 2:** RA from Other Software Vendors and Professional Services Firms
- Consult the respective websites and developer networks.
- Pro: Often close to implementation reality.
- Con: May promote proprietary concepts.

**Alternative 3:** RA from Standards Body or Book Author (e.g., The Open Group)
- E.g., The Open Group is in the process of standardizing a SOA reference model. OMG SysML extends UML to define core SOA concepts. “Enterprise SOA” and “SOA in Practice” come with their own approaches.

**Recommendation:** All listed reference models have their place. Whichever one you choose, make sure to profile to relevant subset and to provide concrete usage examples.
AD Issue #9 – Overall Integration Layer Design Pattern

**Scope:** Entire Integration Layer

**Decision drivers:** Number and technical diversity of service providers and consumers, change dynamics

**Phase:** Solution Outline

**Role:** Lead Architect

---

**Issue: Integration Style (Conceptual Level)**

Which architectural pattern (if any) is selected to structure the integration layer?

Background reading: Hohpe/Woolf “Enterprise Integration Patterns”

---

**Alternative 1:** 

*Vanilla Message Broker or SOA ESB*

- Hub-and-spoke (stateful or stateless) with broker as hub, message channels as spokes
- Pros: Flexible and extensible, can mediate technical differences
- Cons: There is a risk of scope creep – business logic might end up in integration layer.

**Pros:**

Flexible and extensible, can mediate technical differences

**Cons:**

There is a risk of scope creep – business logic might end up in integration layer.

---

**Alternative 2:**

*Peer-to-Peer Integration (No Integration Layer)*

Each service consumer directly talks to one or more service providers, e.g. via file transfer, shared database, or remote procedure invocation patterns.

**Pros:**

Less architectural components

**Cons:**

Many connectors, maintenance and extensibility issues, development and management effort. Consumers and providers tightly coupled.

---

**Recommendation:** Introduce ESB pattern if loose coupling (i.e., location, format, protocol, and implementation transparency) is a valued strategic architectural principle.
AD Issue #8 – A Detailed Integration Layer Design Issue

Decision drivers: Reliability needs, systems management capabilities, availability of provider

Phase: Macro Design
Role: Integration Architect

Issue: Message Exchange Pattern (Conceptual/Technology Level)
Do consumer and provider communicate synchronously or asynchronously?
Background reading: Hohpe/Woolf “Enterprise Integration Patterns”

Alternative 1: Request-Reply
SOAP/HTTP
Simple to manage, but no guaranteed delivery, so *might* have to deal with undelivered and/or duplicate messages

Alternative 2: One Way over Reliable Transport
JMS or WS-RM
Consumer and provider up times can differ; guaranteed delivery (once and only once) when using persistent messages. Must manage dead letter queue.

Alternative 3: Pseudo-Asynchronous
Combination of Alternative 1 on application integration layer, Alternative 2 on underlying transport layer
Same as Alternative 1, but guaranteed delivery

Recommendation: Do not follow an MOM hype – decoupling in time is just one of several dimensions of loose coupling. The equation (NOT RM => NOT SOA) does not hold true.
AD Issue #7 – Integration Layer Technologies

Issue: Integration Technology (Technology Level)
Which technology standards should be used to implement the chosen integration patterns?
Background reading: Pautasso et al “RESTful vs. Big Web Services”

Alternative 1: WS-* Technologies
SOAP/HTTP, WSDL and other appropriate XML languages
Pros: Tool and standards support, interoperability
Cons: Perceived to be complex and under vendor control (tools and middleware needed for XML editing and processing). Limited support in scripting languages.

Alternative 2: RESTful Integration
HTTP with certain principles (see PhD thesis by Roy Fielding)
Pros: HTTP ubiquity, scalability, perceived to be simple
Cons: No interface definition language and few tools, so many integration responsibilities shifted back to application developer. Does not support asynchronous messaging. Not transactional.

Recommendation: Both alternatives have their place. Avoid being biased in the decision making. Note that REST often is positioned as a architectural style (alternative to SOA).
AD Issue #6 – Overall Process Layer Design Pattern

Decision drivers: Business scenario, resource coordination/protection needs, team preferences

Solution Outline
Lead Architect

Issue: Service Composition Paradigm (Conceptual Level)
Should a process layer be introduced? If so, how is service composition done?
Background reading: Leymann/Roller “Production Workflow”

Alternative 1: Workflow (BPM)
Executable process with sub-processes, activities, etc.
Highly expressive, tool and engine supported, separates composites from atomic services
Additional programming paradigm (graph-based)

Alternative 2: Composite pattern via modern programming language (e.g., OO)
Proven, used in other architectural layers
Does not provide engine support for business transactions; runs the risk of not structuring the business logic layer properly (into sub-layers).

Alternative 3: Other Layer
Composition responsibilities may also be taken by presentation layer (mashups) or by integration layer (stateful broker)

Recommendation: Introduce process layer and realize it with workflow pattern and technologies if business scenario is long running multi-actor scenario with advanced resource coordination/protection requirements. Use OO composition in other cases.

BPM: Business Process Management
OO: Object Orientation
AD Issue #5 – Transaction Management Topic

Decision drivers: Resource protection needs, data currency, performance

Macro Design Application Architect

Issue: Invocation Transactionality Pattern (Conceptual Level)

Should a business process, its activities, and the service components it invokes run in a single or in multiple system transactions?

See ICSOC 2007 paper by Zimmermann et al. for available patterns.

Alternative 1: Transaction Islands

Do not share Tx context
Best performance, loose coupling, but no full ACID protection for resources.

Macro Design Application Architect

Recommendation: Use Transaction Islands as default, Stratified Stilts for long running, distributed processes. Decision injection into model transformation or BPEL code in WebSphere Integration Developer is possible.

Alternative 2: Transaction Bridge

Share Tx context
Best resource protection, but large, long running Tx tightly coupling activities and services.

Alternative 3: Stratified Stilts

Use asynchronous messaging and suspend Tx
Supports loose coupling best, but no full ACID protection.

Workflow engine selection

Tx: Transaction

ACID:
Atomicity, Consistency, Isolation, Durability

SCA: Service Component Architecture
**AD Issue #4 – Process Layer Technologies**

Issue: *Workflow Language* (Conceptual Level)

Which BPM/workflow language should be selected?

Background reading: Websites of standards bodies and analyst reports

**Component Model**

**Entire Project or Solution**

Decision drivers: Language expressivity, tool and engine support, portability

**Macro Design Application Architect**

**Component Model**

**Workflow pattern selection**

**Component Model**

**Workflows**

**Workflow Engine Selection**

**Alternative 1: WS-BPEL 2.0 (a.k.a. BPEL) from OASIS**

Standardized, (partially) supported by tools and engines

**Alternative 1: BPEL 2.0 predecessor such as BPEL 1.x, WSFL, or FD(M)L**

Might expose engine-specific features such as dead path elimination better than standard.

**Alternative 3: Other Standardized or Proprietary Language**

Other service composition and BPM paradigms exist, e.g. based on Petri Nets rather than graphs.

**Recommendation:** This decision is often constrained by the workflow engine selection (if that decision is made upfront). Look for standards compliance if portability is an important requirement; avoid usage of proprietary language features during process modeling.

**BPM:** Business Process Management

**BPEL:** Business Process Execution Language
**Issue: Service and Component Layer Patterns (Conceptual Level)**

How should the service and the component layer be organized?

Background reading: Fowler “Patterns of Enterprise Application Architecture”, Buschmann et al “Patterns of Software Architecture”, Gamma et al “Design Patterns”

**Alternative 1:** Traditional GOF, POSA and PoEAA patterns

- Façade, Broker, Domain Model are three of many applicable ones.

**Alternative 2:** Domain-Driven Design (DDD) pattern (Evans)

- Context and Ubiquitous Language patterns are particularly useful.

**Alternative 2:** Core J2EE Patterns

- See book by Alur et al
- Proven for low level design and coding (not all are architectural)

**Alternative 3:** Patterns from Other Languages


**Recommendation:** Follow a best-of-breed approach and select patterns from multiple languages as justified by project requirements and already existing designs. E.g., apply the façade/wrapper pattern to separate the service layer from the component layer. Make services as stateless as possible (Client State or Database State patterns). Identify, make, and enforce pattern adoption decisions.
AD Issue #2 – Service/Component Layer Technologies

Decision drivers: Container services and tools, market acceptance, portability

Macro Design Application Architect

Component Model

 Entire Project or Solution

Issue: Container Technology (Technology Level)
Which interface definition language and container technology is used?
Background reading: Web portals such as InfoQ and TheServerSide.com

Alternative 1: WSDL and SCA (several versions)
See OSOA.org and W3C.org
Expressive, platform-neutral
XML documents hard to create and maintain manually

Alternative 2: Java Enterprise Edition (JEE) and remote object interfaces (e.g., EJB)
See various Java websites (from Java vendors, from others)
Mature, in use since 1990s
Under vendor control

Alternative 3: Other, e.g. Ruby on Rails, Spring framework, and many more
Support for modern container concepts such as dependency injection
Vendor support?

Recommendation: This decision is often constrained by service container selection. Look for standards compliance if portability is an important requirement. Find a balance between container features (services) and maintainability and keeping architectural control.
AD Issue #1 – Addressing Service Granularity Topic

Scope: Service Operation  
Role: Service Modeler  
Phase: Macro Design

Decision drivers: Functional requirements (domain model), capabilities of BPEL, SOAP, WSDL, XML processors (verbosity), interoperability, network topology, number of deployment artifacts and generated code structure, strong vs. weak typing philosophy.

Issue: In Message Granularity (Conceptual/Technology Level)
How many message parts should be defined in the service contract?  
How deep should the part elements be structured?  
The four alternatives have not been published as patterns yet.

Alternative 1: Dot Pattern  
Single scalar parameter  
Easy to process for SOAP/XML engines, much work for programmer

Alternative 2: Bar Pattern  
Single complex parameter  
Deep structure and exotic types can cause interoperability issues.

Alternative 3: Dotted Line Pattern  
Multiple scalar parameters  
Handled by all common engines, some programmer convenience.

Alternative 4: Comb Pattern  
Multiple complex parameters  
Combination of options 2 and 3, biggest overhead for processing engines.

Recommendation: All alternatives have their place; alternatives 2 and 3 are often chosen. Base decision on layer and service type. Avoid overly deep nesting of data structures unless you want to stress test the XML processing. Minimize message verbosity.
Vision: Enterprise-Wide Guidance Model

- Enterprise architecture group owns and maintains guidance model
  - Input comes from solution architects on development/integration projects
  - Quality assured, aligned with enterprise IT strategy

- Does not mandate a particular architecture, but frames design work
  - Recommend certain alternatives:
    - E.g. “always use document/literal SOAP”
  - Ban others:
    - E.g. “no open source assets can be used due to open legal issues”
  - Finding a balance between freedom-of-choice and freedom-from-choice

- Allows project teams to share lessons learned and best practices
  - Actionable enterprise architecture
  - Enterprise architects perceived as friends, not foes
The Power of Process
Coming to Portland in March 2011

SEPG North America 2011 | Oregon Convention Center | Portland, OR

www.sei.cmu.edu/sepg/na/2011
New SEI eLearning Portal Brings SEI Courses to You

The SEI is pleased to announce the new SEI eLearning Portal, a new platform for the development and delivery of SEI eLearning courses to conveniently meet your professional development needs.

The SEI eLearning Portal provides expert instruction as well as exercises, assessments, and other resources, creating a rich educational experience that is accessible by learners worldwide.

- learn at your own pace
- communicate easily with instructors
- access courses 24/7
- study at home, work, or on the road
- read materials online or download for later
- track your course progress

http://www.sei.cmu.edu/training/elearning/
CERT's Podcast Series: Security for Business Leaders

www.cert.org/podcast/
For more than 20 years, the SEI has been at the forefront of software engineering.

By becoming an SEI Partner, you join forces with a software engineering pioneer and an institute whose credibility provides a solid foundation during uncertain economic times.

SEI Partner Network

www.sei.cmu.edu/partners
Do you have the knowledge you need?

SEI Training

www.sei.cmu.edu/training
NO WARRANTY

ANY CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to olz@zurich.ibm.com.

The views expressed are those of the author and do not necessarily reflect those of Carnegie Mellon University or its Software Engineering Institute (SEI).