LogicLibrary™

Best Practices for Software Development Asset (SDA) Reuse

Introduction

Everyone agrees that reuse of software development assets (SDAs) is a "good thing" and that service-oriented architectures (SOAs) require organizations to institutionalize services as reusable assets. Nonetheless, there is considerable debate within IT organizations as to how to "get from here to there" – i.e., how to move from the current state of IT project development, with its emphasis on immediate, time-driven project objectives, to "SOA nirvana" with flexible, loosely-coupled services that are produced from requirements driven by core business processes, meet near-term objectives and facilitate strategic business enablement. Not an easy task, but it is possible to move towards this goal incrementally by viewing development processes from the perspectives of SDA production, distribution and consumption.

Some Terminology

To set the stage for the discussion, let's begin with definitions for SDA and SOA.

SDAs: Knowledge Assets and Executables with Maintainability, Discoverability and Consumability

What is an SDA? Simply put, it is "something of value to an IT organization." This definition is generic for a reason: development assets are so wide-ranging that an all-encompassing definition must be necessarily vague; however, it can be clarified by example. Within the world of SDAs, there are two major types of assets: knowledge assets and executable assets. Knowledge assets consist of information used by the IT organization to do its work more consistently, efficiently and effectively. Examples of knowledge assets include architectures, design patterns, processes, templates, etc. Executable assets are the things most technical people think of when discussing reuse: components, services, APIs, schemas and other deployable packages. SDAs of both types are meant to be reused, so special care must be taken in their production.

©2005 LogicLibrary, Inc.

What makes an asset reusable? For example, does a J2EE component become an asset simply by providing its jar file? Probably not, unless the component's functionality is extremely simple and very obvious. While the deployable jar is a very important work product (i.e., artifact) of the software development process, it does not make the component an asset in and of itself. In order for something to be considered an asset to the IT organization, it must be maintainable, discoverable and consumable.

- ***Maintainability*** introduces such concepts as version control (discussed in more detail below), models and other design documentation, as well as requirements traceability (why the asset was implemented in this way from a technical and business perspective).
- ***Discoverability*** means potential consumers of an asset can find it in a timely fashion – for example, via keywords, domain taxonomies or models to which the assets are mapped.
- ***Consumability*** involves looking at an asset from the point of view of a future project that might use the asset: are a user guide, a well documented API, sample client code and other artifacts available to help the user rapidly understand how to apply the asset to a project? Are dependencies to other assets (and to prior versions of this asset) specified and easily navigated?

The process of building an asset creates metadata that represents the asset – describing the asset from various points of view. This metadata presents a composite view of the asset across its entire development and deployment lifecycle, with indexes (or references) into the various point tools that hold the work products associated with the asset, such as document management systems, requirements management systems, version control repositories, defect tracking systems, test automation tools, etc.

©2005 LogicLibrary, Inc.

SOA Key Concepts: Core Application Functions, Coarse-Grained Services and Message-Oriented Infrastructure

For a concept that has probably reached the peak of its hype cycle, a surprising amount of debate and confusion over what constitutes a service-oriented architecture continues to exist. Much of this debate tends to get stuck in technical details, perhaps because the technology infrastructure most frequently associated with SOA - Web services – is still maturing as standards evolve and organizations like WS-I define preferred interoperational modes of service deployment. In short, conversations about SOA seem to stall at the technical level because Web service technology remains in flux.

Looking beyond pure technology, some core concepts rise above the technical muck to help define the fundamentals of a service-oriented architecture. An SOA provides for the definition and delivery of **core application functions** through a series of **coarse-grained services** meant to be assembled through a **message-oriented infrastructure**.

Enabling Core Application Functions

First, an SOA must support the delivery of core application *functions*. To clarify, this is not a comment about releasing applications. It is a statement about *enabling* a development process that delivers business value by being flexible and responsive so high-quality applications are delivered faster, for better service and competitive advantage. An SOA must yield appropriately decoupled services that can support multiple applications – both end-user (customer, partner, and internal) and machine-facing – without service reimplementation. Services must be reusable without major rework, or they just add Yet Another Layer Of Technology (YALOT) to an already messy technical infrastructure.

Coarse-Grained Services – "Right-Sized" for Application Assembly

Services within an SOA must be sufficiently coarse-grained to enable meaningful assembly of applications. Right-sizing services for this purpose is one of the more challenging issues facing architecture teams instituting an SOA. At one extreme, services which support

©2005 LogicLibrary, Inc.

"get customer's middle name" (for example) are far too fine-grained for easy reuse in application assembly (and any resulting applications built from such services will perform abysmally), while services that implement top-level, end-user-facing business processes, such as "close accounting books for fiscal year," are too broad and completely impractical to implement. Finding the right middle ground of services that can support multiple business processes with appropriate granularity is probably the biggest challenge to an IT organization.

Tying Services Together with a Message-Oriented Infrastructure
Services are of no use if they cannot be consumed within actual applications. Tying them together into business process flows is typically completed through message passing, often with an orchestration engine of some sort maintaining the long-running process state.

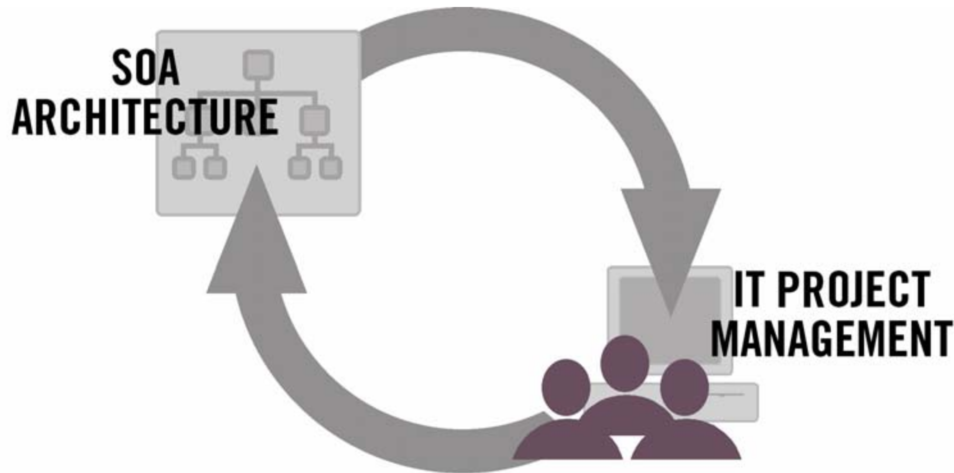
Getting to "SOA Nirvana"

With terminology established, it is time to discuss how an organization can begin to roll out an effective SOA strategy. This conversation does not start with the bits and bytes of SOA infrastructure. Such an infrastructure is clearly necessary as a technology underpinning to support a deployed SOA, but it is insufficient as a guide for SOA definition and development. The most appropriate place to start the dialogue is with proper architectural and project governance, which is required to "put your IT house in order." These processes prioritize the development of services while keeping the broader objectives of SOA in mind. They also ensure that services are built with sufficient security, quality and consumability so that downstream consumers can easily find, understand and register their use of the services in application and business process integration projects.

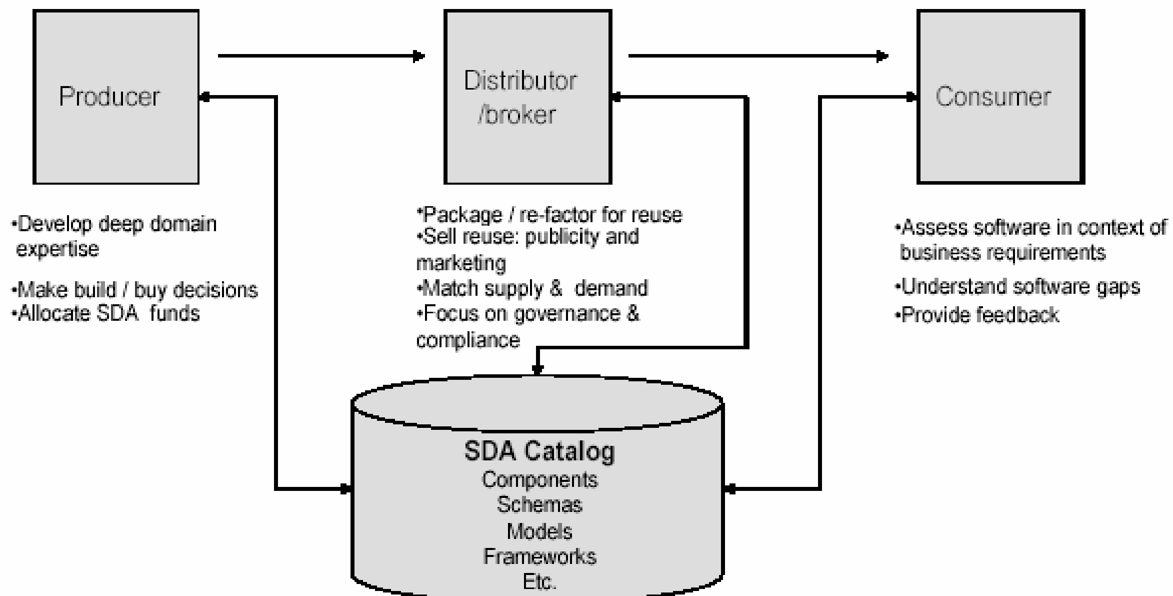
All of these objectives fit under the umbrella of "SOA governance," the means by which enterprise architecture teams oversee IT

©2005 LogicLibrary, Inc.

development projects and IT project teams provide feedback on that guidance.



Within this "big-animal" governance lifecycle, a supporting lifecycle for asset reuse becomes visible: asset production, asset distribution, and asset consumption.



©2005 LogicLibrary, Inc.

The remainder of this paper discusses five best practices within the asset-reuse lifecycle.

Asset Reuse Best Practices

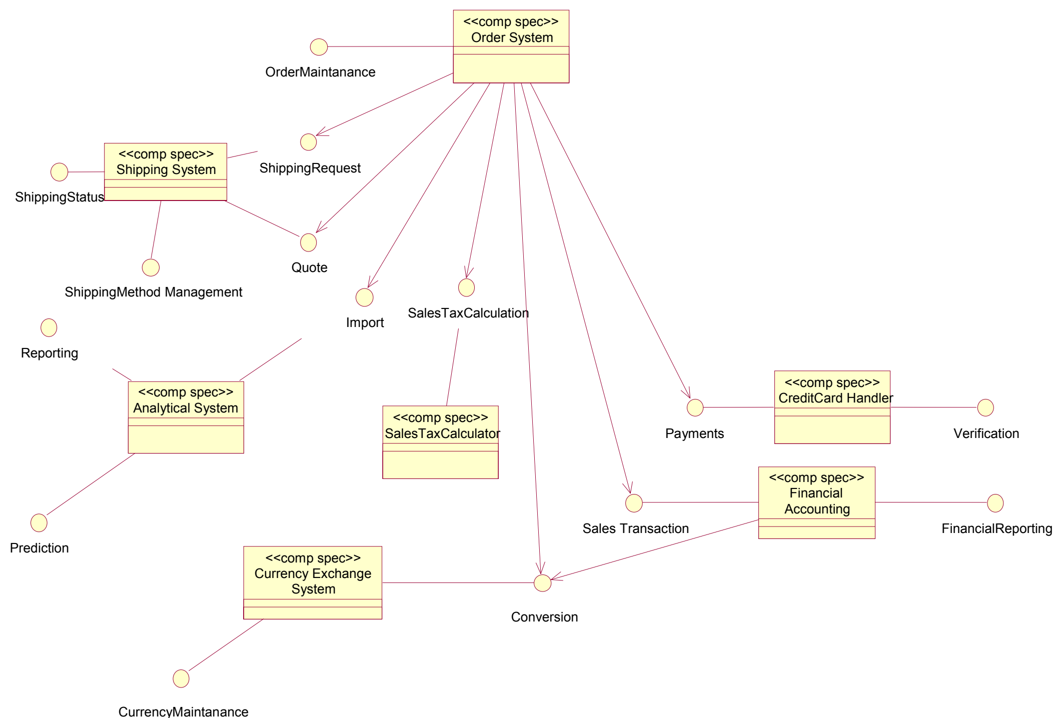
Best Practice #1: Pragmatic Service Interface Modeling and Definition
Going back to the original definition of SOA, one of the key aspects of an SOA is that the services defined and deployed within the architecture must provide core application functions to potential consumers. This is easier said than done. Services within an SOA cannot be effectively developed in a "bottom-up," ad-hoc manner. Bottom-up development of services is inherently driven by immediate project needs – how do I solve this specific problem with a specific implementation (often driven by the influence of existing applications and their behaviors masquerading as true business requirements). What happens when an organization defines and implements its services with this mindset? The service layer simply becomes YALOT – more spaghetti code that implements a monolithic application in a different technology instead of improving business process flexibility.

But, services also cannot be defined solely in a "top-down" manner. Top-down business process analysis left to its own devices often leads to one of two outcomes: "analysis paralysis," continual refinement of a model hoping to reach perfection (which never comes), *or* "Big-Bang" projects that try to "boil the ocean" – defining and implementing everything at once, usually with disastrous consequences ("death march" projects, schedule slips and/or cost overruns).

So where to go from here? The objective is to design and implement business services that support the immediate project's requirements with enough flexibility to meet future business process needs, both projected and unknown. This is a pretty stiff challenge and one that is not likely to be met in a single step. In reality, there must be a pragmatic balance between where the business is and where it needs to go, moving services towards the objective in iterations.

©2005 LogicLibrary, Inc.

These iterations should move the service towards the goal of building a coarse-grained business architecture (or model). The coarse-grained business model should be driven by key business processes (initially, a representative set of high-priority processes). Architects and business analysts should use these processes to extract and define a normalized set of functions that are grouped together based on behavioral affinity (or, in UML terms, components and interfaces) to create a straw-man set of initial, target definitions for the services. This effort creates a roadmap for prioritized service definition and development.



Has the exact set of services and operations been defined as a result of this top-down modeling activity? Of course not, but it has produced a starting point for the real work – the detailed analysis, design and implementation of the services necessary for the current set of prioritized projects. The needed services are identified from the business reference model, based on business process (and project)

©2005 LogicLibrary, Inc.

prioritization, and their definitions are formalized. Ideally, each service definition should be driven by the requirements extracted from at least two separate processes. Otherwise, designing a service based on a single use case will probably result in a fragile, narrowly-defined service that will not be flexible enough to meet the needs of the next set of prioritized projects. Formalizing service definitions may lead to modifications in the reference model, including the identification of new services – which is just fine! This is the first step in iteratively refining the business architecture.

Once the service interface is specified, project teams can proceed with detailed service design and implementation. Often, this means assessing the current set of production applications to understand which aspects of the applications are candidates to support required services, and then combining and re-factoring application capabilities by implementing adapters that provide the necessary glue and compensation logic. Usually, these adapters are implemented behind a component façade which may have been generated from the original service definition, as specified in WSDL (or, alternatively, service operations may have been defined as methods on a component interface, with a WSDL document and service client proxy code generated from that starting point).

Best Practice #2: Production Lifecycle Review Points

As mentioned above, we need to ensure that the defined and implemented services are properly aligned with the enterprise architecture, use correct implementation techniques and technologies, and provide enough supporting information to enable potential consumers to rapidly discover and understand them. How is this accomplished? By applying appropriate review points in the software development lifecycle (SDLC) and defining a *virtual/matrixed SDA architectural review team* to complete these reviews.

©2005 LogicLibrary, Inc.

Who should be on this review team?

- A ***team leader*** drawn from the enterprise architecture organization whose dedicated responsibility is building a successful SDA reuse program.
- ***Matrixed team members*** drawn from participating project teams. The members should have lead designer/developer skills, and their work on this team should be recognized and allocated as 10%-20% of their job responsibility. Assignment to this team should be promoted as a talent-recognition award and a growth assignment for the individuals involved. A rotating membership (perhaps six to 12 months in duration) serves to train younger developers in architectural principles and then allows them to carry that knowledge back to their project teams, increasing the overall skill level of those teams.

Consideration should also be given to including members from the business analyst organization to ensure that the business functionality defined by a service truly reflects the enterprise's requirements.

The primary responsibility of the team is to review services under development. The objectives of this review will vary depending upon the stage within the SDLC to which a particular service has progressed. At a minimum, it is recommended that organizations review services under development at the following points in the SDLC:

- *Requirements complete:* All business requirements are documented and the initial service definition has been specified (ideally as WSDL) so reviewers can validate the service against its business architectural context.
- *Design complete:* The implementation approach has been defined with sufficient documentation (e.g., UML design models completed, relevant legacy APIs identified) to allow reviewers to validate the design against technical and application/integration architectural contexts.

©2005 LogicLibrary, Inc.

- *Implementation complete:* The service has been implemented and deployed in a test environment, with sufficient supporting documentation (e.g., sample client code, automated/manual test cases and test results, usage guide) to enable a potential consumer to understand the service and trust its quality and stability.

Other review points may also be appropriate based on organizational needs and objectives. However, do not overwhelm development teams with process for the sake of process. They will quickly revolt against jumping through seemingly arbitrary hoops. The objective should be "just enough process" to provide appropriate guidance at key points in the production and consumption lifecycles to keep things on track. Just as right-sizing a service may be iterative, finding the right level of process control for the organization may be iterative also. Consider starting with as "light" a process as is feasible and adding more process steps only as needed.

Appropriate tooling can greatly assist organizations in effectively deploying their governance processes. In fact, applying process to product in a pilot project is an ideal way to validate and iteratively refine the process. Industry expert Donald J. Reifer, when discussing how to implement a practical reuse program, states, "…using a pilot project to demonstrate the value added from a well-designed and deployed infrastructure is highly encouraged. Pilot projects force senior managers, middle managers and technologists to determine how to take the technology and use it in concert with the organization's process under real budget, schedules and constraints."[1]

Once a service's review has been completed, the team makes it available to the broader community. An asset metadata library can be of great assistance in this process since it provides automated asset validation and supports the steps in the review process with configurable, role-based approval mechanisms that include notifications and audit trail functionality. After publication to a

[1]Donald J. Reifer , "Implementing a Practical Reuse Program," *Component-Based Software Engineering*, pp. 453-466

©2005 LogicLibrary, Inc.

metadata library, potential consumers are free to discover the service and its capabilities, provide feedback and incorporate the service into their development projects.

It is likely that implementing an SDA review team will have some very useful side effects. Forming such a team can break down organizational barriers, helping to mitigate any potential political issues associated with a reuse initiative. Simply put, creating this type of team can eliminate the "us vs. them" mentality that often occurs when governance organizations and top-down processes are imposed on development organizations. The review team also is likely to discover additional opportunities for reusable services (and enhancements that make existing services more robust and reusable) through their informal communication. This "active discovery" of new reusable capabilities can accelerate the creation and adoption of services within the SOA initiative.

Best Practice #3: Managing Produced Services as Internal "Products"

As services begin to roll out and are consumed within projects, an organization will rapidly reach the point where the next set of business process requirements affects one or more of the existing services. How will the new process be supported while preserving a stable operating environment for existing service consumers? Because services (like components) are meant to be used in more than one application, organizations need to plan for the incremental enhancement of their services over a long deployment lifetime. In effect, organizations planning to build a robust, stable and extensible SOA need to treat their services as "products."

What does treating a service as a "product" mean to an IT organization?

- Each produced service must have a regular and well-defined release cycle. This release cycle needs to occur often enough to meet consumer needs on a timely basis, but not so often as to churn existing consumers. Typically, a release cycle between three and six months is appropriate for most organizations,

©2005 LogicLibrary, Inc.

allowing them to meet new service needs without unduly disrupting existing applications.

- Services must preserve backward compatibility wherever possible. Deprecation techniques provide time to migrate to newer service releases by identifying obsolete operations and notifying existing consumers that the operation will be removed from future releases of a service interface (also known as "end of life-ing" in product management terms). Service providers should offer n-1 version support at a minimum – all services provided in the prior version (except those marked as deprecated) should be preserved intact in the current version. In addition, consider providing a "grace period" where both service versions are supported so consumers can make any necessary changes to integrate the updated service. Dynamic runtime binding techniques via a Web services management infrastructure (e.g., service proxies or UDDI-based late binding) can also simplify the process of migration from an older service to a new version.
- A mechanism must be established by the enterprise architecture team and service-review team to gather new requirements and enhancements for services. Consider establishing a "product manager" role within these organizations to manage and prioritize the aggregate set of business requirements for a service. The product manager should solicit feedback from current and potential consumers of the service, consolidate those requirements and codify them (with the assistance of the enterprise architecture team) for eventual implementation by service development teams.

Treating services as "products" has a clear impact on SDLC tools used within the development teams. Some examples of these impacts include:

- Version Control. Be prepared to establish a source code baseline within the version control repository whenever a new version of a service is released into production (or create a thread label for later use as a baseline, if needed). The

©2005 LogicLibrary, Inc.

service-provider team must be able to simultaneously maintain a production service while developing the next version of a service.

- Requirements Management / Defect Tracking. Organizations need to manage and document requirements and defects against a particular service on a version basis, noting the version against which the requirement or defect originated and the target version for resolution. Most requirements management and defect tracking tools easily support this level of documentation.
- SDA Management. Maintain all "valid" versions of a service within the SDA library. At a minimum, consider defining these lifecycle states for the services:
  - "Under Development" – Only available for the requirements-gathering and planning purposes of the application development team. The service is not available for general development use.
  - "Production" – Generally Available (GA) version for use in new development.
  - "Retired" – Still in use by existing applications, but not available for use by new applications.
  - "Obsolete" – All applications should be migrated from this version. The version metadata is maintained for traceability and audit purposes only.

Best Practice #4: Delivering Web Services to Consumers via an Integrated Asset Metadata Library

Ad-hoc Solutions Are Not Enough

Ad-hoc distribution schemes may be sufficient for managing two or three services used by a small community, and even that is debatable since as long-term traceability is lost if organizations do not maintain service usage records from initial service deployment. However, ad-hoc approaches do not scale to meet the needs of larger SOA and reuse initiatives. Spreadsheets and static Web sites used to distribute services rapidly become out-of-date. Verbal, "call the architect" approaches to communicating knowledge about available and

©2005 LogicLibrary, Inc.

planned services turn critical resources into information bottlenecks. And, why build a repository and distract development teams from their core responsibilities when there is a proven, integrated, enterprise metadata library to buy? Building a home-grown library will also likely delay an SOA implementation and other application development and integration projects.

A Registry Is Not a Repository

UDDI registries are just that – a means of *registering* deployed services to enable operational late binding to one of many deployed service endpoints using a specific service (e.g., for purposes of failover, scalability and geographic distribution). Any minimal repository capabilities in a UDDI registry are a side effect, not its intended purpose. In short, a UDDI registry is not explicitly designed as a repository so it does not provide the functionality of a true repository. For example, a UDDI registry has limited search metadata and cumbersome search interfaces more suited towards runtime look-up than human interaction; is not well suited to managing other SDA types, such as components, legacy application APIs and knowledge assets like design patterns; and is not well integrated into the development environment, particularly integrated development environment (IDE) tools. To summarize, a UDDI registry is not a viable solution for meeting serious metadata repository requirements.

Checklist for an Asset Metadata Library

Ultimately, the objectives in selecting and deploying an asset metadata repository/library are to effectively govern reusable asset production, to make it easy for potential consumers of the assets to find them, and to track asset usage for purposes of change management, impact analysis, and ROI determination. Accordingly, consider these important features when evaluating SDA libraries:

- √ Governed and configurable asset metadata ***assembly*** and ***validation***
  - o Standardized metadata definition
  - o Per-asset-type metadata validation and enforcement
- √ Configurable (manual vs. automatic) asset publication
  - o Newly defined SDAs

©2005 LogicLibrary, Inc.

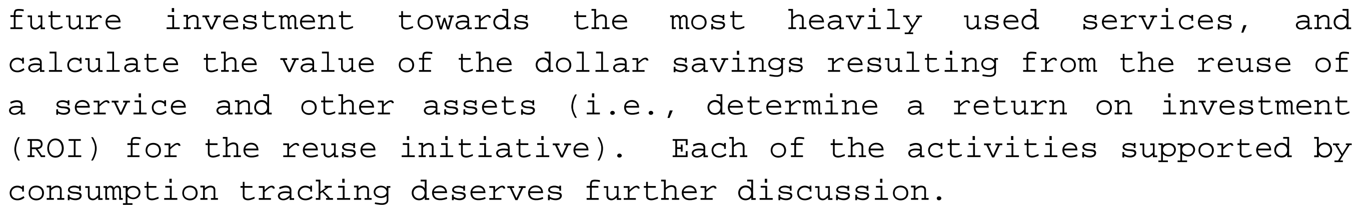
  - Updated SDAs
  - New versions of existing SDAs
  - Organization-defined roles and review/approval processes
- √ Passive and active distribution modes
  - User-based SDA subscriptions
  - Automated search notifications during asset creation/ updating
  - Multiple search modes
- √ Project and asset-specific collaboration
  - Discussion forums
  - Persistent searches
  - Asset notifications
- √ Multiple UI options
  - Thin-client
  - Deep IDE integration
  - API-based integration

Best Practice #5: Service Usage Traceability, Impact Analysis and Return on Investment

Fast forward a bit. Services have been built and deployed that are well aligned with the business architecture and business process needs. They have been published into the asset metadata library so application development teams can easily find them. Now, the first application projects are beginning to use the services. So the reuse initiative is finished? Not quite… One of the key objectives of an SOA is to create a flexible set of reusable services that grows over time to support all of the key business processes. This growth will inevitably result in changes to services that are already deployed (as discussed in best practice #3), making it necessary to understand who is using which assets. In other words, consumption activities need to be scoped and tracked at a project level.

Project-scoped, asset-consumption tracking allows for better control over where services are being used, enabling several critical activities. With this level of tracking, a service-provider team can easily inform consuming applications of service versioning, guide

©2005 LogicLibrary, Inc.



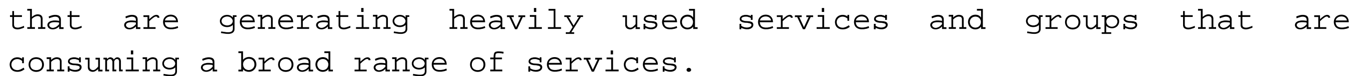
future investment towards the most heavily used services, and calculate the value of the dollar savings resulting from the reuse of a service and other assets (i.e., determine a return on investment (ROI) for the reuse initiative). Each of the activities supported by consumption tracking deserves further discussion.

Depending upon the business, the structure of the development organization, and other factors, an organization may need more or less control over the services being used by various application development projects. At one extreme, an organization may want to encourage broad-based reuse of a set of core services, even going so far as to "pre-register" specific SDAs for a development team to use. At the other extreme, there may be certain assets which are highly sensitive due to privacy concerns, legal or compliance rules, trade secret preservation and other rationales. Limited information may be published about such SDAs, restricting access to sensitive artifacts until the necessary approvals and compliance checks have been completed. The SDA library should support these widely varying consumption models through simple configuration settings.

As discussed in best practice #3, effective versioning of services is key to producing a viable SOA over the long term; however, well managed versioning is meaningless if there isn't a way to easily and automatically inform service consumers when new versions are planned and deployed so they can outline a graceful, controlled transition from the prior version. Traceability through project-based consumption within an SDA library provides the information necessary to proactively manage downstream consumption of versioned services. In addition, early notification allows consumers to participate in the requirements-gathering process for a new version.

Tracking which SDAs are used where quickly creates a picture of where future service development and maintenance are likely to be heaviest. After all, a service used by 10 different mainline applications is more likely to require enhancements and defect fixes than one used by a couple of analytical reporting applications. This information is invaluable to management for resource planning and allocation. It can also be used to recognize and reward service-production teams

©2005 LogicLibrary, Inc.



that are generating heavily used services and groups that are consuming a broad range of services.

Finally, since enterprises are in business to make profits, cost savings are often a major driver for implementation of an SOA or other SDA reuse initiative, especially at the executive "C level." Without the ability to trace asset consumption, it becomes difficult, if not impossible, to quantitatively determine the savings (and resulting ROI) of a reuse initiative. A leading metadata library will provide built-in ROI calculation reports based on proven reuse metrics developed by industry experts like Dr. Jeffrey Poulin[2].

Summary

To be effective, an SOA or other reuse initiative needs to iteratively define the business context. Work with business analysts to define and prioritize the business architecture, mixing top-down analysis and normalization with bottom-up, service-harvesting from existing systems, blending these two approaches based on project priority. Align the existing application inventory against the prioritized business processes that result from this modeling effort. Don't try to "boil the ocean." Instead, pick the key systems that support business needs, combining top-down and bottom-up approaches to service definition and implementation.

As service production starts, define and manage the production, distribution, and consumption of services through a set of well-defined review and approval processes. The right tools, including an asset metadata library, are critical to supporting these processes and, ultimately, delivering quality assets to potential consumers. There is no need to define the perfect governance processes before deploying a metadata library; the processes will never be right the first time. It is much more effective to deploy an "80%-solution"

[2] *Measuring Software Reuse: Principles, Practices and Economic Models*, Jeffrey S. Poulin, Addison-Wesley, 1996.

©2005 LogicLibrary, Inc.

into a series of pilot teams, get feedback and iteratively refine the processes side-by-side with the library.

Treat services as independent "products." Recognize that development teams are no longer in the business of building monolithic applications. They are building "application elements" with the flexibility to support multiple applications and long-lived enough to span multiple parallel versions in development, deployment and obsolescence.

Finally, keep track of where services are being used. Only through such traceability can an organization show how much money has been saved through an SOA or reuse initiative – and rightfully claim its hero status!

©2005 LogicLibrary, Inc.

About the author

Brent Carlson is vice president of technology and co-founder at LogicLibrary, Inc. Carlson drives the development and delivery of LogicLibrary's products. He is a 17-year veteran of IBM, where he served as lead architect for the WebSphere Business Components project and held numerous leadership roles on the "IBM SanFrancisco Project." He is the co-author of two books: *SanFrancisco Design Patterns: Blueprints for Business Software* (with James Carey and Tim Graser) and *Framework Process Patterns: Lessons Learned Developing Application Frameworks* (with James Carey). Carlson is also a frequent presenter at industry conferences, including Web Services EDGE 2005, Software Architecture Summit 2005, Enterprise Architect Summit 2004, Java Pro Live! 2004, Microsoft Tech-Ed 2004, Microsoft PDC 2003, IBM Rational Software Development User Conference 2004, regional user groups and Microsoft Architect Council meetings. He is a BEA Regional Director and was named to *InfoWorld's* 2005 Top 25 CTOs. Carlson holds 16 software patents, with eight more currently under evaluation.

About LogicLibrary

LogicLibrary is the leading provider of software and services that make it possible for enterprises to manage and reuse software development assets (SDAs). The company's patent-pending technology provides a comprehensive and collaborative approach for creating, migrating and integrating enterprise applications for use in service-oriented architecture, Web services and other software development initiatives. Additionally, LogicLibrary's BugScan provides powerful, easy-to-use code-scanning technology that helps architects, developers and IT professionals ensure the highest levels of security throughout the software development lifecycle. For more information, visit www.logiclibrary.com.

©2005 LogicLibrary, Inc.