Twelve Common SOA Mistakes and How to Avoid Them

Yefim V. Natis, Massimo Pezzini

Agility, incremental software engineering, software sharing (reuse) and lower cost of heterogeneous operations are among the promises of a mature service-oriented architecture (SOA). However, achieving these benefits often entails overcoming formidable technical, organizational and political hurdles. Planners must think long term but act in pragmatic steps, focusing on the twin goals of long-term agility and short-term cost optimization. Most importantly, planners must avoid several common mistakes that have been known to derail SOA initiatives in the past.

Key Findings

- Long-term services are best designed systematically, but this systematic approach must be balanced against — and deliver benefits that justify — the added costs associated with planning and quality assurance.

- Organizations pursuing SOA initiatives can improve their chances of success by avoiding the 12 common pitfalls identified in this research.

Recommendations

- Focus on avoiding the proliferation of unshareable services. Reward both reusability and reuse, and establish a center of excellence (COE) to provide guidance and governance.

- Invest in systematically designed sets of fundamental core services. Make the design of services and service interfaces independent steps in software design, involve business analysts early and often, and coordinate service design with data design.

- Don't underestimate the technical challenges of SOA. Despite the relative ease of the initial steps, recognize that large-scale SOA implementations require an SOA backplane and an understanding of key SOA-enabling middleware.

- Don't underestimate the cultural, political and marketing challenges of SOA. Avoid starting too big, and avoid selling SOA to upper management too soon. Understand the diverse objectives that motivate different business audiences, and tailor your communications accordingly.
TABLE OF CONTENTS

Analysis ............................................................................................................................................. 3
  1.0 Introduction ............................................................................................................................. 3
  2.0 Two Critical Measures of SOA Success ............................................................................... 3
  3.0 Twelve SOA Mistakes and How to Avoid Them ................................................................. 4
    3.1 Mistake No. 1: "Irrational SOA Exuberance" ................................................................. 5
    3.2 Mistake No. 2: Forgetting the Data ................................................................................ 6
    3.3 Mistake No. 3: Leaving SOA to the "Techies" ............................................................... 7
    3.4 Mistake No. 4: Succumbing to "Not Invented Here" Syndrome .................................... 8
    3.5 Danger No. 5: Starting too Big ....................................................................................... 9
    3.6 Mistake No. 6: Starting in the Wrong Place ................................................................. 11
    3.7 Mistake No. 7: Assuming That Everyone Thinks Like You ........................................ 12
    3.8 Mistake No. 8: Choosing Dictatorship to Combat Anarchy ....................................... 13
    3.9 Mistake No. 9: Underestimating the Technical Issues ............................................... 14
    3.10 Mistake No. 10: Allowing Unshareable Services to Proliferate .................................. 15
    3.11 Mistake No. 11: Excessive Centralization ................................................................. 16
    3.12 Mistake No. 12: Selling SOA Before You're Ready .................................................. 18
  4.0 Conclusions and Recommendations .................................................................................. 18

Recommended Reading .................................................................................................................. 19

LIST OF FIGURES

  Figure 1. Balancing Costs and Agility ....................................................................................... 4
  Figure 2. Service Interfaces Should not Equal Software Interfaces ..................................... 5
  Figure 3. Degrees of Service-Data Normalization ................................................................. 6
  Figure 4. The "Not Invented Here" Syndrome Is Contrary to SOA Principles ......................... 9
  Figure 5. The Service-Oriented Application Enterprise .......................................................... 10
  Figure 6. Potential Starting Points for Service Design ............................................................ 11
  Figure 7. Three Styles of SOA Governance .......................................................................... 13
  Figure 8. The Technical Complexities of a Large-Scale SOA Environment .......................... 15
  Figure 9. Linking SOA Domains Through an Enterprise SOA Backplane ............................ 17
1.0 Introduction

SOA is the central theme of most modern software initiatives, although some companies remain skeptical about its ultimate benefits. Agility, incremental software engineering, sharing (reuse), lower cost of heterogeneous operations — all these are part of the promise of SOA, and all can pose formidable obstacles.

We recommend that the approach that proved successful for software initiatives be applied here as well: Think long term, but act pragmatically and insist on frequent, tangible milestones and measurable results. This research examines this approach and other keys to SOA success, and it highlights some crucial dangers to be avoided along the way.

2.0 Two Critical Measures of SOA Success

The critical, long-term objectives of SOA and key measures of its success should focus on two areas — agility and cost.

- **Agility**: The degree of agility achieved in enterprise IT, and, consequently, in the enterprise's core business, is critical. This agility is measured in the time to market for the development of new services, as well as for the re-composition, change and removal of services inside or outside process sequences. In an agile IT context, all these activities are reasonably quick and minimally intrusive in running the business application environment.

- **Cost**: Greater expense does not necessarily lead to greater agility (more is not always better). Consider the cost of SOA in two dimensions: the cost to deliver a new service and the cost of change in the established SOA environment. The relationship between cost and results is not linear in these efforts.

Systematic design and management efforts are essential to support reuse, service isolation, cohesive functional representation, impact control and change — all of which are prerequisites to agility. As systematic investment increases, so do the costs — but agility increases as well.

After some continuing systematic investments, the cost per service of adding a new service or of changing an established service begins to decline as the organization is well-empowered and the methods well-understood by participants. As the tools and procedures improve the productivity of the engineering process, the incremental cost of engineering reaches a low endpoint, while the agility reaches the high point — at which point, the system achieves perfect balance (see Figure 1).
If the enterprise continues to adopt new procedures and processes to gain further control and automation, then it may begin to lose this balance. If so, then costs will begin to increase again (reflecting the growing complexity in the environment), but agility will begin to decrease (reflecting the growing overhead in the system). At some point, excessive controls will become counterproductive, and agility will decline.

Excessive overhead of methodologies, consensus gathering, multiple levels of review and approval (all good in modest doses) can bring the SOA environment to a halt. Making changes to the systems becomes increasingly problematic and time-consuming, so systems become cumbersome and inflexible. Engineering teams then begin to avoid the process. Eventually, nothing can be produced, the platform is quietly ignored and abandoned, and the SOA initiative fails.

**Action Item:** Design long-term services systematically (at the added cost of planning and quality assurance). But recognize that short-term services don’t require investment in systematic qualities (and will prove expensive and inflexible if kept active during a long period).

### 3.0 Twelve SOA Mistakes and How to Avoid Them

These are a dozen of the most common mistakes Gartner has observed in SOA implementations:

- Irrational SOA exuberance
- Forgetting to consider the data
- Leaving SOA to the "techies"
- "Not invented here" syndrome
- Starting too big
• Starting in the wrong place
• Assuming that everyone thinks like you or in the same way
• Choosing anarchy or dictatorship as leadership styles
• Underestimating technical issues
• Allowing unshareable services to proliferate
• Excessive centralization
• Selling SOA before you’re ready

3.1 Mistake No. 1: "Irrational SOA Exuberance"

The easiest way to expose "services" to the outside world is by generating externalized interfaces from established class definitions in Java or C#, using interface definitions in Java Enterprise Edition (Java EE) or Common Object Request Broker Architecture (CORBA), or communication area definitions in IBM's Customer Information Control System (CICS) Transaction Server. Such transformations are supported by many tools and can be entirely automatic.

Although this may be an easy approach, it's also a poor one. It causes a range of problems and doesn't result in true SOA.

Services in SOA aren't just any software modules. Although services are implemented in software, they must be defined to reflect the partitioning of the application's business functionality, not the technical partitioning of software. In a well-functioning SOA environment, there are always far fewer service interfaces than component or object-class interfaces (see Figure 2).

Figure 2. Service Interfaces Should not Equal Software Interfaces

![Figure 2. Service Interfaces Should not Equal Software Interfaces](image)

Most pre-SOA objects and components have been designed to optimize the operations and engineering of software; thus, they don't meet the SOA requirement of business design. Objects and components remain integral aspects of software engineering, but the design of SOA-style services must be conceived separately.

Most SOA services are designed for a basic, interactive SOA environment. The design of event processing is very different from the design of interaction. The consideration of behavior (event-driven or interactive) must be applied when the services are being designed — which is quite separate from generating programming interfaces.
Excessive numbers of services — that is, those that can't be readily matched to the business model of the application — are a sign of a “check-mark SOA” environment. Such environments may feature repositories full of services, volumes of documentation and an impressive collection of new tools and middleware, but these environments also provide no agility, incremental software versioning, reuse or benefits of true SOA.

**Action Item:** Make the design of SOA services an independent and dedicated step in the software design life cycle. Design these services as externally facing business functions, not as technical software modules (even though the services are implemented as technical software modules).

### 3.2 Mistake No. 2: Forgetting the Data

Services in well-designed SOA environments are long-term assets. Services designed without systematic planning may work well for short-term opportunistic projects, but these services are poor candidates for the demands of changing environments in the long term. The process of systematically designing a service model resembles that of designing a data model. In both cases, the impact is long term, and the normalization of the designed elements is a sign of maturity and quality. Although the principles of the normalization of services are different from the normalization of a data model, an important principle of service normalization is its relationship to the underlying data. Forgetting the data in the process of the design of services easily can result in services that deliver poor performance and can challenge the integrity of the application.

The objective of the normalization of service design is eliminating redundancy, preventing "white spaces" (functionality that was missed in the initial design and that must be opportunistically patched in later) and delivering a business-meaningful partitioning of application functionality (so that the collection of offered services is meaningful to the potential outside "reusers"). A systematic approach to the normalization of the service-data relationship enables the coordination of service design and the design of the underlying data model. A systematic method also establishes a level of "commitment" of the service to the data of record with which it works (see Figure 3).

**Figure 3. Degrees of Service-Data Normalization**

![Degrees of Service-Data Normalization](image-url)

Source: Gartner (October 2007)
• Anything goes — Service implementation uses any data that must be accessed.

• Ownership — Service implementation distinguishes "its" data and accesses all other data only via programmatic (service) interfaces.

• Encapsulation — Service implementation prevents other application software from accessing "its" data directly and instead offers programmatic interfaces for this purpose.

• Object — Service implementation takes data ownership to the point where its master value is known only inside the service implementation. (Direct database look-up no longer provides a meaningful view of data — only the service interface calls do.)

Making the transition from no normalization to the advanced normalization of service-data relationships is a process of increasing the maturity of SOA. Systematic efforts always should begin with advanced normalization. Opportunistic efforts might proceed from lower to higher architectural levels of as SOA tools, skills and levels of confidence improve. Some higher levels of normalization may prove to be unrealistic, given the enterprise's performance and management realities.

Just as the data catalog and data design are administered by a dedicated role — a database administrator (DBA) — the services catalog and design administration also should be made the responsibility of a designated role: a service registry and repository administrator (SRA). As with the DBA, the SRA manages the consistency of the catalog and enforces the guidelines that protect against redundancy, proliferation and unauthorized modifications of the service catalog.

**Action Item:** Strive to design services in a manner that is coordinated with the design model of the services' underlying database.

### 3.3 Mistake No. 3: Leaving SOA to the "Techies"

One promise of SOA is to narrow the divide between the business and IT sides of the enterprise. Because this isn't the objective of any single software project, the benefit easily can be missed when evaluating the quality of an SOA design. However, in the long term, creating a closer link and a better understanding between the business and the IT organization can be the greatest progressive contribution of SOA to the agility and competitiveness of the business organization.

To keep the focus on this objective, the best SOA initiatives involve business analysts and software designers in a collaborative effort early in the process, and establish the organization and processes needed to ensure that the collaboration between these two sides of the enterprise is constant and routine. Such an approach may encounter resistance because it may be perceived as expensive and intrusive, so it requires encouragement and enforcement from the leadership of the organization.

When the collaboration between business analysts and software architects is successful, the resulting SOA projects have increased levels of reuse, the IT organization delivers new solutions to the business faster and the cost of change is lower. In the longer term, the IT organization develops greater business expertise, and the business develops a greater understanding of the IT process. This outcome, in turn, improves the quality and synergy in the groups’ independent work.

When the SOA process is left mostly to the IT side of the organization, services risk being designed to optimize software performance and ease of use, not to reflect the business functionality of the application. Clarity of business-semantic content of the service interfaces is essential for cross-application integration, composition or multienterprise use.
**Action Item:** Recognize that SOA design is a shared challenge for the business and IT sides of the enterprise, and organize your SOA practices accordingly. Bring both sides to the "design table," or risk creating an underpowered and underused SOA environment.

### 3.4 Mistake No. 4: Succumbing to "Not Invented Here" Syndrome

One of the most anticipated benefits of SOA is the increased reuse (sharing) of software. Just as object classes are reused (often via inheritance) inside Java or C# software files and "remotable" components are reused (often via RPCs) inside the applications, the success of a service can be measured partly by the degree to which it is reused by outside applications. (It is important to note that SOA with no reuse is still a progressive architecture, because it delivers a clear software design and enables incremental software development, deployment and maintenance.)

Although SOA's benefits aren't disputed, reuse faces many challenges. For example:

- It can be difficult to design interfaces to address external requirements that the applications' original designers weren't prepared to support.
- Not every application has internal information resources that are of general-enough interest to be reused.
- The security, integrity and performance characteristics of different applications vary, and mismatches can block reuse.

In addition, cultural obstacles can derail an SOA reuse effort. Many IT organizations suffer from a "not invented here" syndrome, in which programmers, project leaders and architects don't trust the other teams, or simply want to develop entire solutions by themselves. Not only does this syndrome cause practical problems, such as redundant programming efforts, overstaffing, or lost opportunities from lack of available resources, but it also poses a major obstacle to the success of an SOA reuse initiative.

In a mature SOA environment, the catalog of available services includes the software for the particular project, other projects in the same IT division, projects from other divisions of the enterprise, purchased business applications, software from other (partner) enterprises and Web-based service interfaces (see Figure 4).
The IT environment must develop a culture where these external solutions are understood and used where applicable. The effort to expose services for reuse when they're not paired with the incentive to reuse other applications' services is likely to fail.

**Action Item:** Reward the reuse of software designed by others. Foster a technical and cultural environment where such reuse is considered a characteristic of excellence in software engineering and preferable to custom programming.

### 3.5 Danger No. 5: Starting too Big

Many enterprises, especially those that believe they're late in using SOA, tend to leap from SOA skepticism and a wait-and-see strategy into a sudden, strategic commitment. The question of "why use SOA" is replaced with "why aren't we there yet?" Budgets suddenly are available, and results are awaited eagerly. In such circumstances, the IT organization often is encouraged to start thinking large scale and to begin building a new, long-term infrastructure.

Developing a long-term vision from the start is a useful undertaking — as long as it preserves the agility to continuously adjust the vision based on real results. However, plunging into large-scale SOA development efforts without considering scalability and manageability issues typically is a dire mistake.

A large-scale SOA environment manages hundreds of business services organized into functional groups in one dimension and application sets in another (see Figure 5).
These services include new, old and very old software that has been adjusted in various ways to work together, as well as purchased software, software as a service and other externally sourced interfaces. Hundreds of processes and transactions may use and reuse services and may demand security, accountability, integrity and performance across the invariably heterogeneous and often multienterprise transaction span. Pre-SOA systems can be quite complex, and SOA can multiply the potential points of failure by breaking up a large monolith into multiple service implementations and reducing the capability of middleware to maintain control of the end-to-end transaction context. SOA reuse also increases the dependence of one application on another, further complicating management efforts (another reason to evolve toward greater levels of maturity in SOA — gradually).

All this complexity requires a well-developed discipline of design and management. For most organizations, this discipline emerges amid the process of learning and organizing; it cannot be acquired by signing consulting contracts or by purchasing infrastructure technologies.

Because SOA is a long-term, complex initiative, enterprises should invest in developing the required understanding, best practices and organizational culture before committing to mission-critical SOA projects. Leaping into such a computing environment is treacherous. Thus, gradual adoption is imperative for most enterprises.

In the case where an organization with minimal experience in SOA is engaging in a large software project and is interested in using SOA principles, we do not advise avoiding the project for lack of established practices. Rather, we recommend that the large project be subdivided into smaller components (in line with now-prevailing, agile project management techniques) so that the SOA effort is applied initially in a relatively small scope to be expanded over time. Early SOA projects should not last longer than six months from the start of design to the delivery of results.

**Action Item:** Recognize that starting too big can lead to big mistakes. Think strategically, but act tactically. Develop a long-term vision for SOA, but implement it incrementally, learning during the process and managing the risks of transition.
3.6 Mistake No. 6: Starting in the Wrong Place

The question of where in the business cycle to begin designing services is a critical one in SOA. The immediately obvious answer is to follow the intended first users (see Figure 6).

**Figure 6. Potential Starting Points for Service Design**

For example, if the service is requested by a user-facing application, then you might design a service to match the data requirements of the user interface. If the service is required to fulfill a step in a business process sequence, then you might design it to fit the data requirements of the steps in the business process. This way you will end up with as many services as user interfaces and steps in the business process designs — often leading to a redundant and ever-growing collection of services.

This answer works only in its own context. If you are concerned only about a particular process or user interface (which is legitimate if the requirement is temporary and no long-term impact is anticipated), then the service can reflect only those requirements. If, however, you are concerned about the longer-term health of the application or about achieving the benefits of SOA agility and reuse, then thinking this narrowly won't help. A more-consistent, more-systematic (and more time-consuming and more resource-consuming) approach is to design a cohesive set of services around the application’s business model or data model.

In some cases, you can design a set of services around business models and data models. The data model can be used to encapsulate the business data, and the business model can be used to link the business analysis of the application with its software implementation. In this example, the business-based services typically consume the data-facing services, and the latter often end up being internal components never directly exposed to the outside world.

In other cases, the data model is the sole defining model for the application, or data may be encapsulated within the business services. The opportunistic services, which fulfill the current requirements of user interfaces and business process designs, should use the available
systematic services (built to reflect the business and data models of the application) to preserve internal integrity while responding to urgent, opportunistic initiatives.

Data and business models are the best choices on which to base systematic services, user interfaces and business process designs. However, for the rapid opportunistic service design, the choice of middleware must never be an influence in the design of the services. Instead, this choice should follow and support the established design of services. (This includes consideration of the required quality of service and productivity at the appropriate cost and complexity levels.) Users should delay the choice of dedicated SOA middleware until their service topologies are established and the requirements for the type and depth of the middleware can be established properly.

**Action Item:** Rather than attempting to take a shortcut to opportunistic, quickly created and soon-discarded services, invest in systematically designed sets of fundamental core services as the initial stage of design, allowing for rapid opportunistic extensions later.

### 3.7 Mistake No. 7: Assuming That Everyone Thinks Like You

An SOA initiative is a long-term endeavor. Its success must be measured based on multiple criteria and schedules. Every level and role in IT and business organizations is likely to have a different understanding of SOA and distinct success criteria for SOA investments.

SOA isn't one thing for all people. Having originated as a technical design pattern for advanced distributed systems, it has become a subject of interest beyond the programming community. Although some common expectations of SOA are shared among all parties — especially that it will bring the benefits of increased agility — different players in an organization judge the success of an SOA initiative using different criteria. For example:

- **To a programmer,** SOA is a form of distributed computing in which the building blocks (services) may come from or be offered to other applications. SOA increases the scope of impact of a programmer's product and adds to his or her resources, but it also closely resembles familiar modular software design principles.

- **To a software architect,** SOA translates to the disappearance of fences between applications. Architects turn to the design of business functions rather than to self-contained and isolated applications. The software architect becomes interested in collaboration with a business analyst to get the clearest picture of the business functionality and scope of the application. SOA turns software architects into integration architects and business experts.

- **For CIOs,** SOA is an investment in the future. Expensive in the short term, its long-term promises are lower costs and greater flexibility in meeting new business requirements. Reuse is the primary benefit anticipated as a means to reduce the cost and time of new application development.

- **For business analysts**, SOA is the bridge between them and the IT organization. It carries the promise that IT designers will better understand them, because the services in SOA reflect the business functions in business process models.

- **For CEOs,** SOA is expected to help IT become more responsive to business needs and to facilitate competitive business change.

**Action Item:** Manage the expectations of SOA investments by understanding that the involved parties don't all envision the same outcome as the objective. Consider these differences in tailoring business communications at all levels.
3.8 Mistake No. 8: Choosing Dictatorship to Combat Anarchy

Throughout the years, individual IT projects, groups, divisions and domains have developed a hunger for independence. The prevailing experience with enterprisewide methodologies, standards and guidelines has been a largely negative one, involving increased costs and uncertain benefits. Recognizing that most large-scale initiatives don't last very long, many departments have developed a habit of stalling and waiting for each "wave" to subside. SOA often is introduced and perceived as a large-scale initiative and, thus, faces a similar attitude.

In fairness, the independence of projects and departments is often in the interest of larger IT objectives. It enables architecture and technologies to be selected that are best-suited to the needs of different projects and skills to be organized for the most-effective run toward a project's objective. However, the SOA benefits that can be achieved by an isolated project or department are limited. The larger the scope of a coordinated SOA effort, the greater the potential benefit of the increased agility, integration and sharing of resources.

The full independence of projects might be seen from the perspective of the whole organization as "anarchy" (self-governance by citizens), making it impossible to impose shared objectives on the larger organization (see Figure 7).

Figure 7. Three Styles of SOA Governance

![Figure 7](image)

A drastic alternative to such anarchy is dictatorship, wherein departments and projects are denied independence and are forced to follow central command on most matters. Shared objectives are communicated easily, but individual areas operate well below optimal levels, unable to choose the direction that best fits their individual requirements. Neither anarchy nor dictatorship offers the balance required for a successful SOA environment.

The two conflicting requirements — independence and interdependence of individual IT projects — can be met through the enforcement of a respectful, coordinated effort that includes the representation of all interested and affected parties. The "brick walls" separating projects can be replaced with "glass walls," thus preserving some independence and enforcing some coordination.

A well-organized SOA environment always includes an SOA COE. The role of the COE is to involve all participants early and to facilitate all necessary coordination between the otherwise independent projects or divisions of the organization. The role of the COE also is to minimize unnecessary intrusions on the independence and internal processes of the participants. This way
the projects and divisions can preserve their independence while contributing to the shared, larger objectives of the organization. Some might call this organization model "federalism."

A successful SOA environment fundamentally depends on intelligent levels of motivated coordination among largely independent projects, departments, divisions and other affected organizational units. Well-functioning SOA environments will be perceived as an attraction rather than a threat, thus eliminating much of their historically inherited isolationism.

**Action Item:** Establish a COE as the center of coordination among SOA projects. Ensure that its directives are minimally intrusive and that it recognizes the realities of individual projects.

### 3.9 Mistake No. 9: Underestimating the Technical Issues

The technology issues associated with SOA are more challenging than vendors would like users to believe. Web services technology has turned SOA into an affordable proposition for most large organizations by providing a universally accepted standard foundation. However, Web services play a technology role only for the SOA backplane, which is the software infrastructure that enables SOA-related interoperability and integration. Web services support interoperability — within and beyond the boundaries of the enterprise (for example, business-to-business integration) — although standards such as Web Services Description Language (WSDL) and Business Process Execution Language (BPEL) also facilitate some aspects of application development.

Service implementations must be developed and executed on platform middleware (such as transaction-processing monitors or enterprise application servers) providing the necessary quality of service (performance, availability, scalability, manageability and security). Frequently, however, service implementations also must be carved out of pre-SOA applications using integration tools such as adapters, programmatic integration servers or other forms of integration middleware. Service consumer applications must be deployed on platforms that support multiple devices and presentation styles. Web services protocols must be complemented by reliable, high-performance and secure protocols to support the most-demanding requirements. But, at times, Web services are simply overkill, and developing simple, custom, application-specific protocols based on plain-old XML (POX) over HTTP is an adequate approach.

Setting up the SOA infrastructure entails making technical choices regarding the software platform: application servers, portals, integration suites, business process management (BPM) tools, transaction-processing monitors, integration servers, communication middleware, policy management and enforcement tools, and XML appliances all play a role in large-scale SOA initiatives (see Figure 8).
To make appropriate choices, users must understand the complex world of middleware. Despite SOA's growing popularity and the availability of proven, SOA-enabling middleware, the risk of making wrong decisions looms large for newcomers.

**Action Item:** Use point-to-point Web services connections only for small-scale, experimental SOA projects. If the number of services deployed grows to more than 20 to 30, then use a middleware-based intermediary — the SOA backplane.

### 3.10 Mistake No. 10: Allowing Unshareable Services to Proliferate

A well-designed SOA environment minimizes the number of services needed to support consumer applications. A typical SOA goal is maximizing the number of shareable services (that is, services that are engaged by more than one consumer application). Shareable (or reusable) services yield faster development or integration of consumer applications, lower development costs and easier maintenance. Organizations shouldn't expect 100% of their services to be shared across more than one application. As an example, some services provide functionality that is unique to one application. However, sharing doesn't happen by chance — it requires governance, incentives, discipline and tools.

If the average number of services per consumer application significantly exceeds 20 — or if less than 10% of services are shared — then these may be signs that the amount of services sharing is suboptimal. The reasons for this can be multiple and include:

- **Service duplication** — The same business functionality has been implemented many times, with only minor variations.
- **Overly granular services** — The services implemented are too fine-grained, and to complete a business transaction, it is necessary to invoke a myriad of them.
• **An overspecified SOA environment** — A number of unnecessary services have been implemented.

To avoid an excessive proliferation of the number of services and to maximize sharing in the context of an SOA initiative, support the service definition phase by a formal process involving business analysts, architects, developers and integration specialists. The goal is to ensure that a service meets the broadest possible set of requirements (current and, possibly, future). Such a process is costly and lengthy. Therefore, only services with a reasonable chance of being reused across multiple applications should go through validation from development teams other than the team involved in the implementation of the primary application that requires the service. (Even in well-managed SOA initiatives, only 30% to 40% of implemented services are invoked by more than one application. Some services are specific to an application, and there is no possibility of cross-application sharing.)

Document consumable services in a publicly available shared service registry to enable architects and developers to retrieve information about services in production or under development, as well as the associated consumer applications.

To dodge the pain of going through the service definition process, developers who are under pressure for delivery might be tempted to focus only on their short-term issues and to avoid submitting potentially highly shareable services to the process. Therefore, reward developers for their expertise in delivering reusable services and in reusing already available services, not for their ability to produce large numbers of services. In some cases, as an example when organizations look for aggressive IT cost reductions, the responsibility for finding shareable services is assigned to a "service-chasing team" whose goal is to scan the enterprise for potentially reusable services, seek interested "buyers," develop the relevant business case (for example, highlighting the cost savings associated with decommissioning redundant software components that can be replaced by a common, shareable service), and implement and document the service.

**Action Item:** To maximize the reuse and sharing of services, enforce a formal services definition and validation process, implement a design-time service registry, create incentives for service reuse and sharing, and assign responsibility for "chasing" new, reusable and shareable services.

### 3.11 Mistake No. 11: Excessive Centralization

Implementing an enterprisewide SOA initiative is a Herculean effort because of the many political, organizational and technical issues to tackle. For example, business units or subsidiaries already may have implemented their SOA backplanes on the basis of different technology platforms. Or, one department may be committed strategically to a specific packaged application that carries a vendor's own implementation of the SOA principles. Different business units may have different performance, scalability and protocol support requirements for their SOA backplanes. Or, an acquisition can import a different SOA implementation (based on different technologies, standards and governance processes) from the acquired company.

A sound enterprisewide SOA strategy, moreover, must recognize that not all services are equal:

- Some services make sense only in the context of a specific application system (private services).
- Others may be of interest to multiple application systems within a business unit or department (local services).
- A relatively small number of the overall services deployed in an organization (probably no more than 10% to 20%) will be meaningful across multiple organizational units in the enterprise (public services).

Rather than imposing one enterprisewide SOA backplane, one service registry and one set of governance processes and standards, it might be more practical and politically smoother to adopt a federated approach. In this case, the enterprise's SOA initiative is split into "SOA domains" (for example, subsidiaries, business units or departments). Each domain is managed by one business owner (defining the business and functional requirements) and by one technical manager (responsible for implementing the applications and services required), has its own specific SOA backplane and service registry (holding private and local services for that domain), is supported by a domain SOA COE and is managed on the basis of domain-specific governance policies.

Domains can be "federated" through an enterprise SOA backplane and registry (holding the public services). The enterprise SOA backplane is an infrastructure — typically implemented as a subset of a full-fledged SOA backplane, providing transformation and routing services — that links the individual domain backplanes through gateways, transformation tools, appliances, enterprise service buses, or even point-to-point, message-oriented middleware or Web services-based connections. The enterprise SOA backplane and registry are managed by a central SOA COE that establishes common, cross-domain governance, processes and technologies whenever possible (see Figure 9).

**Figure 9. Linking SOA Domains Through an Enterprise SOA Backplane**

Source: Gartner (October 2007)

**Action Item:** If you have a large organization with semiautonomous business units and subsidiaries, then consider using the federated approach to SOA as a way to overcome political, organizational and technical hurdles.
3.12 Mistake No. 12: Selling SOA Before You’re Ready

Small-scale, experimental SOA projects don't require huge investments in technology or sophisticated skills. As long as individual project leaders or architects are willing to risk adopting a popular, although internally untested, approach, introducing SOA will be relatively easy. The success of these initial projects often stimulates SOA adoption by adjacent application areas and larger developer communities. At this stage, the chief technology officer and the architecture team must buy into SOA because more-sophisticated skills and technologies — such as enterprise service buses, service registries or BPM suites — are needed.

At the CIO level, however, SOA often continues to be perceived as a specific approach adopted by a circumscribed set of projects that don't require careful monitoring by the CIO. Often, it is only when SOA proves effective in many business-critical initiatives that CIOs will perceive it as a strategic option — one that will provide enough evidence of benefits to justify the additional investments needed to expand the SOA initiative's scope at a business unit level or beyond.

The strategic, enterprisewide adoption of SOA probably is beyond the empowerment level of most CIOs. It requires the CEO’s — or even board-level — support, because of sensitive organizational and political issues (for example, cross-business-unit cost allocation and governance) must be worked out. However, seeking top management's commitment to enterprisewide SOA too early can be dangerous. Most enterprise initiatives aren't mature enough for this step: Gartner forecasts that, through 2010, fewer than 25% of large companies will have developed the technical and organizational skills needed to deliver enterprisewide SOA. If an immature approach is prematurely exposed to the direct scrutiny of impatient business leaders, then even partial failure to deliver the anticipated business benefits would undermine the credibility of the SOA approach and could potentially threaten the future of its IT sponsors.

**Action Item:** If your organization has adopted SOA only recently, then avoid pursuing enterprisewide SOA initiatives. Instead, focus post-introduction efforts on smaller-scope initiatives, such as intradomain or business-unit-wide projects.

4.0 Conclusions and Recommendations

Agility, reuse and heterogeneous operations are all part of the promise of SOA. But to deliver on this promise, planners must focus on the goals of agility and cost optimization and must avoid the common pitfalls known to derail SOA initiatives in the past. Our recommendations include:

- Focus on avoiding the proliferation of unshareable services, but recognize that sharing doesn't happen by chance — it requires governance, incentives, discipline and tools. Reward the reuse of software designed by others inside and outside the usual developer community. Establish a COE that is minimally intrusive and recognizes the realities of individual projects.

- Recognize that although reuse (sharing) is a valuable opportunity promised by SOA, reuse is not SOA's only benefit. Apply SOA to deliver on other distinctions as well: the clarity of software design; incremental software development, deployment and maintenance; and an increased affinity between business modeling and the design of software. These benefits apply even if SOA is pursued by a single application project in isolation.

- Recognize that large-scale SOA implementations require an SOA backplane — as well as an understanding of and investment in middleware — and that initial SOA efforts may require no new infrastructure or tooling. In the early stages, concentrate on developing best practices for the design and management of SOA-style projects. For the long term,
plan to invest in dedicated infrastructure and productivity tools to ensure best results from SOA efforts.

- Invest in the systematic design of sets of fundamental core services before opening the application to the creation of rapid, opportunistic services. Make the design of SOA service interfaces an independent step in software design, and coordinate this activity with the design of the underlying data model.

- Don't underestimate the political dimension of implementing SOA. Understand that different audiences are motivated by correspondingly diverse objectives, and tailor business communications accordingly. Where possible, use a "federated" approach to SOA to overcome intractable political, organizational and technical hurdles.

- Don't start too big or sell SOA to upper management too soon. Think long term, but act pragmatically and gain maturity through smaller-scale projects, with frequent milestones and metrics along the way.

- Ensure that business-modeling analysts are involved in the design of services and that the services reflect business functionalities rather than the technical partitioning of software, where appropriate.

- Promote a culture of sharing and collaboration throughout the organization. SOA facilitates collaboration on multiple levels, but an organizational culture that encourages isolation and the hoarding of information cannot exploit this opportunity and is likely to defeat SOA.

RECOMMENDED READING

"Applied SOA: Transforming Fundamental Principles Into Best Practices"

"Evaluating IBM, Microsoft, Oracle and SAP Commitment to SOA Governance"

"Findings: Why You Should Bother Building an SOA"

"Five Principles of SOA in Business and IT"

"Flow Management in SOA: One Size Doesn't Fit All"

"Magic Quadrant for Application Infrastructure for Composite-Application Projects, 2Q07"

"Magic Quadrant for Application Infrastructure for New Service-Oriented Business Application Projects, 2Q07"

"Service-Oriented Architecture Overview and Guide to SOA Research"

"Succeeding With Multiple SOA Service Domains and Disparate ESBs"