WHITEPAPER

BEST OF: SOA WORST PRACTICES

PROGRESS
SOFTWARE

TABLE OF CONTENTS

THE BEST OF THE WORST

How many times have you heard: “Security for your service-oriented architecture [SOA] should be tight enough to be secure, but not so rigid as to impact flexibility or performance?” Did you think: Hey, how do we measure this?

You’re not alone. These and other SOA best practices offer good advice and value, but applying them to your business can be tricky. After all, best practices don’t always relate to your specific situation or business model.

The problem goes deeper. Too often, what is billed as a “best practice” turns out in, um, practice, to deliver less than hoped.

This is why Progress Software created ***SOA Worst Practices***. We collected observations of some of the most common yet most easily committed SOA bloopers we’ve seen in the field. We knew that sharing best practices was important—we all do that—but we also realized that knowing what not to do when building an SOA can be just as valuable. ***SOA Worst Practices Vol. 1*** focused on tactical and strategic aspects of implementing service-oriented architectures: employing SOAP standards, the limits of XML firewalls, Web-services reuse and so on. Hot on the heels of ***SOA Worst Practices, Vol. 1,*** we published ***SOA Worst Practices, Vol. 2,*** and zeroed in on the worst practices of organizations that attempted—or attempted to avoid—SOA ***governance***.

STRATEGY, TACTICS, GOVERNANCE: IS THAT GREAT IDEA YOUR WORST PRACTICE?

Last we looked, a Google search for “SOA best practice” returned 4.8 million hits. That’s a lot of advice. Embracing the philosophy of “less is more,” this paper highlights our top 10. (Both volumes of the unabridged versions of these “worst practices” are available from our website at www.actional.com/resources/whitepapers/.)

In this guide, you’ll learn about real-life (and really lame) mistakes made by your peers.[1] We’ll share with you the thinking behind these ideas, why they failed, and better approaches that can help you achieve a more successful outcome. (In other words, you will be smarter after reading this. Really.)

Perhaps you are considering adopting an SOA. Or maybe you are already far down the path of your first. Either way, you will want to keep this guide handy. There’s bound to be a worst practice here that you (or a “friend”) were tempted to try.

If Progress can save even one developer from the SOA worst practices hall of shame, then writing this guide was well worth the effort.

[1] Although we changed the names of all companies (except for ours) mentioned in these worst practices, their stories are true and painful.

THE 10 WORST SOA PRACTICES

We had a lot to choose from but narrowed it down to the top 10 worst practices to date. Something tells us that this is just the tip of the iceberg...

10. GET INSTANT SOA—GRAB YOUR EXISTING APPS AND WRAP 'EM UP!

An SOA doesn't have to be complicated; however, it's important to have a clear SOA strategy and to understand the security risks and ways to mitigate them. Or you could be looking at a recipe for disaster.

The Idea

Granger Machinery, a manufacturer of heavy and industrial machinery, thought that it had a great idea for starting an SOA. The company planned to use Web services in the form of a "wrapper" to make its existing inventory and customer database available to other lines of business. The SOA was initiated by allowing a client to send complete SQL statements in the request to the database. However, this protocol left the relational database susceptible to SQL injection attacks, whereby attackers bypass the SQL statements defined on a Web server in order to inject their own statements.

The IT team thought that it could reduce this risk by using an anti-injection attack feature in its existing security product. This feature would detect and sanitize SQL statements that were written to perform destructive operations, such as "Drop" or "Delete."

Why It Wasn't So Smart

By itself, this Web service approach of putting a wrapper on an existing application or database is not dangerous.

However, if you think about this case, it is actually a tightly coupled approach and requires that the requestor know the implementation details to make the SQL call. This approach suffers from two major problems:

1. It lacks business context. The IT team needs to associate a business context—or rules—around the Web services, instead of allowing "dumb" SQL requests. Such rules define who has a right to put in a request to the SQL database and in what situation they can access data. Individuals would then need to be authenticated before accessing the database. This would prevent the database from blindly giving out information, as well as prevent destructive requests from directly accessing the table.
2. The SQL injection problems are caused by two different issues: bad data that gets injected into SQL statements, and bad SQL statements themselves. The injection protection capabilities described earlier cannot solve these problems.

A Better Approach

By applying SOA and Web service standards, you can quickly achieve reuse and integration goals. However, this approach needs to be part of an overall SOA strategy, or you risk experiencing problems at the application level.

In the case of Granger Machinery, the company needed to address the following in its SOA strategy:

- Who should have access to the data?
- How should they access it?
- Can and should we prioritize the requests and transactions?

Bottom line? Wrapping a service is an excellent way to get reuse out of applications that are already delivering value to your business. It is therefore a good way to build an SOA, so long as you have a business strategy to dictate which services to wrap and the rules that govern their use. Randomly wrapping services, however, can lead to security and performance problems—inside and outside the organization.

Also, remember that building an SOA is an incremental process. You can start by wrapping services that are working well today and rebuild others over time. In addition, you can prioritize what to rebuild and what to build from scratch based on the new business opportunities that come your way.

Example: Problems at the Application Level
If different services are allowed to talk to each other, an unauthorized individual can pull information or use more of the service functionality than you had intended. Imagine that Granger

Machinery had a human resources portal and suddenly different lines of business wanted to use that information, even though they were not authorized to do so. Such a free-for-all could put the company at risk of violating security and privacy laws, as well as slow down performance.

Governance should ensure that such issues are accounted for and built into the strategy. Policies then will perform the control, with management and run-time governance ensuring these polices are enforced and measured.

9. FORGET ABOUT CLONING HUMANS, CLONE YOUR APPS!

Cloning and reconfiguring existing applications might sound like a good way to save on maintenance and development costs. But in practice it is far from a perfect science.

The Idea

The IT team at Jesper Travel, an online travel service, had a novel approach to systems management. It created new applications and new business processes by reconfiguring and cloning existing applications.

One application, for example, had 30 “separate” instances in production, all customized to a certain degree with 50% to 60% of the application core residing in each instance.

The IT team thought that this approach would reduce development time, make maintenance easier, and reduce problem resolution time. They reasoned that:

- More developers would be familiar with a larger percentage of applications.
- The architecture was simple: the two points being the cloned application and the customer.
- It would be easy to plan and compute future costs, since a new customer = new hardware + cloned application + a few modifications specific to that customer.
- A new customer would be a self-contained “business model,” with all the costs associated with it and no shared infrastructure to depreciate.

Why It Wasn't So Smart

This approach was flawed in four major ways. First, if there was a bug in the core code, it was replicated. Second, some of the core code was altered during the customization process, but these changes were rarely documented. So, when the triage team needed to determine why something was not working, it spent far more time doing detective work than actually fixing the problem.

Third, the result of this cloning was redundant infrastructure, maintenance, and development—not only at a business area level, but also at the IT level. The IT team had no way to share capacity between customers, lower the TCO, or manage SLAs. (But the hardware and software vendors were quite happy with this model, as every new customer for Jesper Travel meant additional hardware and software purchases.)

Fourth, core code changes impacted every customer who relied on these services. Following every change, the core and related customized modules had to be tested before the company could redeploy these services. In addition, the IT team could not simply patch applications, but had to go through more steps, all of which translated into a longer release cycle. Plus, this approach embedded rules, or policies, as code into the application—rather than abstracting them. This added to the level of redundancy and made change even more difficult.

Example: Embedding vs. Abstracting
Instead of abstracting one set of security rules that applied to all customers, such as "all customer data must be encrypted," Jesper Travel had this rule embedded in each application.

This greatly complicated systems management. Abstracting would have allowed Jesper Travel to change the rules simply by changing the configuration of the core infrastructure functionality—instead of the code of each application.

A Better Approach

If you want to save on maintenance and development costs, focus on reuse. Unlike cloning and reconfiguring, reuse enables growth within your business, while minimizing complexity within your IT systems.

By building an SOA, you can attack redundancy at the application and infrastructure levels. For instance, you can distill from hundreds of duplicated applications a small set of core application feature sets. These reusable services can be used globally, across applications and business processes, resulting in greater operating efficiency and reduced maintenance costs. In addition, you can standardize, break out (abstract), and control the core infrastructure functionality of key business processes, including security, identity, profile, message queuing and brokering, translation, directory service, etc.

With an SOA, you also can construct new business solutions far faster than by cloning applications. As a result, you can spend more time customizing solutions for each customer, while still leveraging your core set of services. These smaller services actually can help reduce your hardware and software costs as well, since they allow your company to better utilize what it already has. In addition, you can measure, manage, and secure these services or change them easily with policy changes.

8. WE EVEN LET GRANDMA USE OUR WSDL

Reusability is a key benefit of SOA, but context must dictate how services can and should be reused. As this worst practice illustrates, security, privacy, and compliance issues can surface when the use of a Web services description language (WSDL) is not governed.

The Idea

Venton Automotive, an OEM of sunroofs, had provided its human resources (HR) team with a great new tool. The company had designed its internal HR portal so that employees could enter and manage their own information, including skill sets, phone numbers and locations, managers' names, etc.

Likewise, HR validated this information and manually removed records for any individuals who were no longer with the company. The portal functioned largely as a trust-based system, performing only minor data validation, such as validating reporting structure, titles, etc.

Soon the portal was the only place where up-to-date and complete employee profile information could be found. In fact, it was the only place where HR could locate a current organizational chart. Over time, other departments discovered the portal—and began to use it. The audit team, for example, used the information to ensure that everyone in a group had been trained on compliance and privacy issues. And the company telephone operator was able to ensure calls were routed to the appropriate individuals.

When other developers requested access to the HR portal information, the IT team shared the WSDL used to define the service's formats and protocols. The IT team was comfortable sharing the WSDL, for it reasoned that the point of an SOA was to have a reusable service. The main developer shared the WSDL with three or four people, who then shared it with other developers. After a few months, no one was certain how many people were using the service.

Why It Wasn't So Smart

The IT team soon learned that someone had put the WSDL into a library and then shared the library with all the development teams. They soon learned that the Web service had more than 30 consumers. The IT team had no idea that so many users and lines of business were being supported.

They also didn't know:
- Were the right departments being supported?
- Was the data protected or encrypted?
- What was the capacity of the service?
- If it failed, what would be the ramifications?
- What if the IT team needed to version the service?

To make matters worse, people were using the library in production, so the IT team had a development server supporting production applications. This arrangement posed serious security risks and jeopardized adherence to compliance and privacy laws.

In sum, IT management was exposing its systems to substantial risk, while missing the opportunity to show measurable value of an SOA, as demonstrated by reuse.

A Better Approach

To achieve secure reuse of services via an SOA, you want to have a detailed process in place. Such a process must:

- Identify the consumers and providers that depend on the service
- Ensure that the right security measures are in place, IT policy is enforced, and business rules are applied
- Set up management and runtime governance technology
- Validate that the service is running as designed and meeting the business objectives for which it was designed Include a warning mechanism and discovery capability, so you can know immediately when production operations go awry or something slips by the process—such as the inappropriate sharing of a service or WSDL

Bottom line? Sharing a WSDL may seem like an innocent act, but, if not controlled, countless individuals can gain access to a valuable service, putting your data at risk.

Progress's technology, which automatically discovers all services in use, revealed the number of consumers of this Web service.

7. HACKERS WILL NEVER TOUCH OUR DATA, WE HAVE EXTERNAL SCHEMA VALIDATION!

Securing your systems against hackers is a constant battle. Invoking external schema validation may sound like added protection; however, this approach can actually leave you more vulnerable to attacks.

The Idea

Academic Business Services (ABS), a chain of schools specializing in academic degrees for working adults, wanted to increase security for its Web services. It decided to invoke external schema validation. This setting allowed the schema "check" to reside in an alternate location, thus reducing the risk that hackers would be able to decipher the schema using carefully constructed "probing" transactions.

Why It Wasn't So Smart

Often the location of the external validation can be deciphered from the transaction response. A hacker quickly discovered that he could spoof ABS' IP and change the location of the company's schema check. He then initiated transactions with an alternate schema and had it verified by the faux schema validation at the spoofed IP address. This caused legitimate transactions to fail.

So, although hackers could not obtain important data from ABS, they still succeeded in impacting its business.

A Better Approach

You can safeguard your schema by following a few simple steps. First, give the schema to your partners, since you can directly contact them. Then put schema validation on an internal table that is secured by both traditional perimeter defenses and end-point security.

6. OUR SOA INITIATIVE IS A SUCCESS—JUST LOOK AT THE NUMBER OF SERVICES WE HAVE!

In this era of ROI, most companies want metrics to demonstrate that their SOA initiative is progressing and delivering value. So, how do you accurately quantify SOA value? Hint: Don't simply count the number of Web services.

The Idea

Rimstar Bank's executive board had requested a report on the value of its SOA initiative. The bank's CTO, however, was unsure how to quantify this value. Ultimately, he decided to base this figure on the total number of services that the IT team had developed and that were in production. The company had 25 services in production, with five being reused for multiple business processes.

Why It Wasn't So Smart

The primary business value provided by an SOA is reuse. The number of Web services therefore has nothing to do with the success of the SOA initiative. In fact, a high number of services can be a leading indicator that the initiative is failing, since it's likely that reuse is not happening.

At Rimstar Bank reuse was low, with 20 of the 25 services in production still being used as individual applications. Thus, existing applications being converted to services, or newly created services, were not the right ones to tally for an ROI report. After all, they were not being reused by any other applications and processes, nor were they being reused by other lines of business.

A Better Approach

When calculating the value of an SOA, it's best to focus on reuse among existing Web services, rather than the number of newly developed Web services. Reuse is not only a key benefit of SOA, but also something that you can quantify.

You can measure how many times a service is being used and how many processes it is supporting, thus the number of items being reused. This enables you to measure the value of the service. With a little work, you can calculate the cost savings for each instance of reuse, including saved architecture and design time, saved development time, and saved testing time.

Moreover, reuse means fewer services to maintain and triage. So reuse generates savings, and frequency of use drives value

5. GOVERNANCE, AS A PROCESS, CAN BE "CHECKED AT THE GATES"

It would seem logical, on its face, that governance mechanisms could be designed, associated with individual services, and implemented upfront—and that IT could then assume that their role in governance is complete. Despite the apparent logic of this governance approach, it almost inevitably leads to trouble as policies and procedures change over time—in unexpected ways.

The Idea

Amalgamated Electronics is a multinational consumer-electronics manufacturing firm. Amalgamated's IT team, like many others, knew that the success of their SOA would depend on the governance of its Web services. The team spent many hours interviewing individuals in the company to understand the policies that would need to be created. But, the team's governance focus was too static; there was an assumption that the architects could design in governance up front and that everything in future phases—development, test, deployment and runtime—would remain the same. Simply put, the plan was that governance would work as built. Period.

Clearly Amalgamated's IT architecture team was confident of its plans. Or perhaps they simply had unusually great faith in the universe's sense of fair play.

In addition to writing code, it was the developers who were tasked with making sure services complied with defined company and regulatory policies. And it was during the development phase, therefore, that developers built the dependencies between Web services and corresponding policy enforcement (outlined by architects in the design phase) on a service-by-service basis.

So what was wrong with this approach?

Why It Wasn't So Smart

The fundamental problem with Amalgamated's approach to Web services governance was that it assumed that nothing on its services network would change. Or, at minimum, it assumed that change would be simple to track. Their applications, as designed, might pass their governance criteria, and work well at the time they handed them over to the development team. But that's where the trouble began.

Amalgamated's IT team had no visibility, beyond how they designed things, into how the governance measures affecting its applications were implemented, tested, deployed—or how they fared in runtime. Nor, of course, did they have any clear notion of the "change" relationship between what was designed and what was running on the network over time. So, while Amalgamated's developers may have "designed in" governance that worked at first—as policies inevitably changed and services that weren't initially designed were deployed, IT found that its design-phase, service-by-service approach proved too brittle a form of governance to last. Soon enough, compliance policies and other forms of governance on the network began to break.

Bottom Line

The Amalgamated IT team should have asked themselves the following key governance questions:

- How do we know how change will affect what's already there? In other words: it's one thing to design something we know will remain static, but when we have an environment that we know will be subject to change over time, are we able to understand and manage the effect of this change on our deployed SOA—in a reasonably efficient manner?
- How do we know what we don't know—if we don't know it in the first place? Once past the design phase, how do we know what is going on? Are there checks and validation in place after development, testing and

deployment? Do we have visibility into the runtime environment and the ability to ensure that governance policies are enforced?

What Amalgamated's IT team didn't see was that designing for SOA governance, then releasing into a changing, ungoverned world is a recipe for disaster. As regulations change, for example, Amalgamated's out-of-date policies and associated enforcement approach placed the company at risk of fines, loss of revenue—even jail time.

A Better Approach

Only by approaching governance from the perspective of the entire "Service Lifecycle" can organizations help to ensure success. Success can only be achieved when there is a consistent application of governance rules and policies across all services, from the initial design to the actual runtime environment. Without this focus, companies are at risk of not enforcing measures that they intended to, and of not having the visibility to ensure that only services that are in compliance are deployed. Put another way: Web services governance is about minimizing *security* and *risk to the enterprise*. And non-compliance with corporate policies as well as various government regulations—including Sarbanes-Oxley, HIPAA and others—represents some of the greatest of these risks.

Complementing design-time governance with runtime governance offers the user a "layer of abstraction" between the services themselves and the governance mechanism: centralizing management of policies and processes so they can be controlled automatically. It is this centralization that permits the automation of policy enforcement and which prohibits the violation of compliance "behind the backs" of business and IT. In addition, several related concepts and procedures need to be in force:

- Policy enforcement should be decoupled from actual application development and deployment.
- Policies should be independent of applications; they can and should change independently.
- Separate roles and responsibilities should exist for policy creation and enforcement: application developers are not policy experts, and can't develop application logic efficiently when burdened with this task.

4. AS LONG AS I DON'T KNOW ABOUT IT, COMPLIANCE DOESN'T MATTER

In the past—half a decade ago, even—IT organizations could afford to be less responsive to the world around them. The demands from business units for instantaneous analysis of applications and changing rules and policies simply didn't exist. And the regulatory environment—which now imposes sometimes burdensome compliance demands on IT—has changed dramatically, forcing organizations to move with ever-greater speed and attention to processes and policies.

The Idea

Trafalgar Retail Stores is a profitable regional chain of department stores. Recently, however, the company's IT team found itself in some hot water because some services deployed on its network weren't taking into account the latest regulations.

With regulation on the rise in the areas of financial reporting, health care, privacy, environmental policy and countless other fields, firms are being pressured as never before to comply with a dizzying array of complex rules. And there's no end in sight. One would think this regulatory onslaught would have placed the creation of a strategic Web-services compliance plan at the top of the list for Trafalgar Retail Stores. Yet Trafalgar chose to take an old-fashioned, reactive approach to managing its compliance issues—updating their compliance only with the next scheduled upgrade of services.

Why It Wasn't So Smart

As part of Trafalgar's aggressive e-commerce initiatives, the company had been allowing customers to apply for store credit cards online. This application process required the firm to gather sensitive customer information—including Social Security numbers. Recently, evolving privacy regulation began to require that that all Social Security numbers captured online must be encrypted—and even though Trafalgar's IT team was aware of this regulation—they elected to put off the implementation of the Social-Security encryption algorithm until the scheduled versioning of their services at the end of the quarter.

In the interim, hackers managed to gain illicit access into a production database, where they were able to download some of the unencrypted Social Security numbers, along with customers' associated personal data. The resulting cases of identity theft not only hurt the affected Trafalgar customers, but ultimately impacted Trafalgar as well, which ended up suffering substantial financial losses and bad press resulting from a series of expensive law suits.

Trafalgar found itself out of compliance in this particular case fundamentally because of its "defensive" approach to compliance. The IT team's attitude toward compliance ran something like the following:

- "We hadn't heard about *that* regulation ..." or
- "We'll just fix those services or applications in our next revision, when and if compliance violations show up"

Essentially, the IT team had a "see-no-evil, hear-no-evil" approach to compliance on its service network.

The most important thing to realize on this topic is that compliance really does mean compliance. It's not o.k. if you '"missed the memo" or *really meant* to address it. Lack of awareness or a lackadaisical approach to enforcing up-to-date compliance is simply not an excuse—and will not shield your organization from business loss, litigation, fines or even jail terms.

A Better Approach

Adopting a serious stance on governance and the inherent process and technologies that can support it, through the entire service lifecycle, is a requirement in industries where changing governance policies, and a failure to react and update governance measures in a timely fashion, can have serious repercussions. Updating policy "when you get the chance" simply doesn't cut it in this day and age.

Trafalgar would have been better served by abstracting their compliance policies from the application service logic itself—avoiding the need for time-consuming and cumbersome updates to services each time their policies changed. Deploying a comprehensive, automated SOA management solution—facilitating an *active* approach to compliance—would have allowed policies to be updated in a seamless fashion, independently of the services they apply to. This approach would have also provided visibility into the runtime environment, ensuring that the most current policies were both in place and being enforced appropriately.

3. DIVIDE AND CONQUER: APPROACHING SOA SECURITY, MANAGEMENT AND GOVERNANCE SEPARATELY

Company after company today is discovering the secrets of service-oriented architectures: how SOA increase IT agility, cut process costs, boost revenues and drive more dollars to the bottom line. In this enthusiastic environment, some companies are trying to take the "fast track" to SOA. We'll see what happened when one company, in order to leverage a SOA in a highly competitive industry segment, decided to approach SOA by dividing the work into three, parallel task forces. After all, this approach would certainly get their SOA "to market" faster, wouldn't it?

The Idea

BlueSky Tech, Inc., a designer of wafer-handling systems for semiconductor OEMs, operates in a highly competitive market. In order to maintain their industry-leading market share, they looked optimistically toward a SOA to ease their ability to adapt and improve the computer systems and applications used to order and customize their wafer-handling systems to particular OEMs' specifications.

BlueSky's IT team had heard a lot about the advantages of service-oriented architecture. And while at first they didn't know much about the intricacies of SOA, they definitely understood the details of their own business—and why it seemed to be driving them toward the adoption of SOA. Their competition was gaining momentum and they felt the need to operate in a more agile fashion to avoid being overtaken.

So, when the subject of SOA was initially taken up within IT team meetings, it was greeted with enthusiasm. IT management had done its research and had come to the conclusion that SOA was the most effective path toward IT agility and alignment with business needs. They were so optimistic that a move to a SOA needed to happen sooner rather than later, that they began to brainstorm about how they could get there as quickly as possible.

Based on their initial research, senior IT management decided that one of the most critical components to their success in moving to a SOA was having a plan for SOA governance.

The next (obvious) step, once the decision to engage in planning and architecting BlueSky's SOA was approved, was to decide how to approach the overall project. Of particular interest was how the BlueSky IT team would approach their SOA governance plan.

In the end, the IT director took what—on its face—might seem an entirely logical approach: one that is sometimes associated with the Harvard Business School (though it's unlikely HBS has trademarked it!). Faced with a complex problem, BlueSky's IT director did what many an IT director in her place has done in the past: she broke up the problem into logical, constituent parts—and formed corresponding task forces to solve the resulting smaller problems.

For the governance plan, she created task teams with the following missions:

- **SOA Security:** Focus on who has permissions to access services, how the services could be used, and the data-integrity issues associated with those services
- **SOA Management:** Determine how to manage services in the runtime environment and provide service-level agreements (SLAs) to users

- **Governance Policies and Processes:** Focus on capturing the design-time element of the SOA

With three teams working on these problem subsets, the IT organization could arrive more rapidly at a more effective overall governance solution. Sound sensible?

Why it Wasn't So Smart

BlueSky's IT team was thinking in the right direction when it identified the various issues to be addressed as part of governing a SOA. That said, even though it might seem logical to split up responsibilities and plans into individual components—especially as a time-saver to speed up the adoption of SOA—this generally doesn't work as well as one might expect.

When people think about building a SOA—if they consider such things as security, management and design-time governance policy separately—they will inevitably find, too late (after the individual analyses are done), that each of these areas will often overlap and impact others in unexpected ways. While the design-time and runtime components of governance are both essential, if they are not well synchronized (as was the case with BlueSky) governance efforts will fail.

The following examples serve to illustrate the points above:

- BlueSky's security task force devised what they thought were the comprehensive security policies and systems required to secure and control access to their ordering applications. What they didn't focus on (which the governance task force did—given the access to corporate policy information they had) was that BlueSky's clients in the defense industry interact regularly with the Federal government and have more stringent security requirements than their other clients and partners. By the time this fact was accounted for, the security architecture had already been essentially designed—and additional work was required to accommodate this requirement.
- The task force implementing BlueSky's approach to managing the runtime aspects of the SOA planned to support the provisioning of SLAs to the users of BlueSky's ordering systems. What they didn't know about, however, was the existence of a corporate policy that imposed larger fines if the SLAs for certain categories of customer and partner orders were violated during the last week of each quarter. Additional standby services and intelligent routing policies needed to be in place to support this requirement, not to mention additional servers that hadn't been ordered. If the efforts of this group had been more closely aligned with the governance policy group, this setback might have been avoided.

A Better Approach

It's important to take a top-down approach to SOA-governance design and implementation. Realize, for example, that approaches to security, privacy and even key business requirements most likely will be connected—and could impact each other when it comes time to enforce them at runtime.

The bottom-line recommendation here is not to avoid task forces when planning your SOA. The point, instead, is to take a broad, enterprise view of critical topics such as SOA governance, while making sure task forces meet regularly to exchange ideas and remain in synch. In this way, the ultimate governance solution you deploy will be a functioning, unified whole.

2. NO WORRIES: WE KNOW WHO'S USING THE WEB SERVICES THAT COME WITH OUR APP

Increasingly, ERP, MRP and other enterprise packaged applications are being delivered with built-in Web services. These applications include Oracle, SAP and Salesforce.com. While these built-in Web service offerings can be a benefit to the organizations that buy these packages, they can present distinct management challenges.

The Idea

Not long ago, PC MicroCenters, a regional computer and electronics retail chain, upgraded its SAP software. Like many modern enterprise packaged applications, SAP now features convenient, built-in Web services designed to enhance the utility of the application. These services are popular with end users in large part because they are so easy to employ. As it happened, PC MicroCenters' IT team was looking forward to employing these services, too.

Service Rollout Plan: Authorization and Capacity Planning
SAP had always remained under the strict control of IT at PC MicroCenters: in the past, business users typically received the access that IT in general deemed manageable and appropriate. To accomplish this IT used traditional provisioning and authorization mechanisms. This way, IT knew "who was who" and "who was using what".

But as far as IT was concerned, the new Web services that came with SAP would only be used by an IT-defined set of users (and within IT guidelines)., And, perhaps not surprisingly, they took the same approach as in the past—authorizing defined sets of IT users and doing capacity planning based on that narrow group of people.

Why It Wasn't So Smart

Unlike traditional application environments—which can prevent a user from accessing an application entirely, or aspects of an application's capabilities, based on that user's identity—in this case, the built-in SAP Web services were entirely visible and accessible to every business user on PC MicroCenter's network who had an SAP login and password. Unhappily, IT had failed to take this new reality of Web services into consideration.

As it happened, a few of these services proved very helpful (and ultimately, too tempting) to some business analysts in the organization. A number of these analysts found some of the Web services features too enticing to resist. They realized that they now had the ability to do mass downloads of data to Excel (for reviewing weekly sales, shipping, commissions and pricing information). In a number of cases, SAP crashed as a result of the load the analysts were placing on the system—until, finally, IT was able (quite by accident—by means of a hallway discussion) to locate the analysts and put an end to their, albeit well-intentioned, back-channel access.

A Better Approach

Surveys of IT organizations seem to indicate that many of them would like to simply ignore SOA for now—or at least put it on the back burner. In the past, this might have been a sensible approach for some of them. Indeed, many organizations have no crying need for SOA. But the unavoidable reality is that the big-name packaged applications which are embedded in corporate networks around the globe—SAP, Microsoft, Saleforce.com and the rest—are now all coming out with SOA built in. Which means that, like it or not, IT organizations have to be ready: with a SOA strategy and with a governance approach in mind.

In PC MicroCenter's case, a comprehensive SOA governance solution could have immediately told IT who was mysteriously consuming those SAP services—and automatically taken action to shut down the unauthorized "rogue Web service" usage before SAP or any other element on the network could have been affected.

1. NO THANKS: WE HAVE OUR OWN STANDARDS AND PROCESSES HERE

Should governance, in the case of large, "silo-ed" organizations be parsed among separate development teams? Certainly, as this case demonstrates, there would seem to be a logic to this approach. Or should governance be "institutionalized" centrally within the organization? In fact, as we shall see, without such institutionalization, even smaller firms are taking compliance-related risks and are likely wasting resources by replicating their development efforts and especially their attempts at governance on the network.

The Idea

While the bigness of big organizations may sometimes shield them from certain problems routinely faced by smaller firms, size brings with it its own daunting challenges—particularly when the firm is attempting to get its arms around SOA and SOA governance. In the last section, we saw how defense contractor RayStar Industries suffered from serious IT/business-related communication problems. Though RayStar is a successful and profitable enterprise, it suffers from other meaningful pains common to organizations of its size. One of those issues is how it approached establishing (or even beginning to think about) governance across its organizational silos.

To review, RayStar is an extremely diversified company: providing defense components to DoD contractors as well as aerospace products to industry. The company's very separate operating units design and manufacture such things as components for radars, detonators, missile launchers and civil aircraft avionics.

What was not mentioned in the last section is that RayStar's IT/business relationships are even more complicated than previously portrayed. The company doesn't just have one "overarching" IT organization serving the entire enterprise. Instead, each of the several business units has its own IT team—each with its own CIO.

With this broader perspective on the IT/business "organization" at RayStar, one can almost picture the kind of organized chaos that reigned amongst the various business-unit IT teams there. Really: picture it. Each development team within each separate business unit would build their versions of, say, shipping or ordering or inventory services—to suit their own product lines. Because, after all, the shipment processes around satellite components wouldn't be remotely similar to those for detonators, right?

And so it went. As far as the separate business-unit development teams were concerned, there was no basis for talking to the other guy. Each "local" development manager and architect was king of his own hill. And, this made sense, to a degree. One guy's standards just didn't seem to apply to the other. Those missile-launcher-ordering-process guys could just go ahead and keep their specialized services as far as the avionics-ordering-process guys were concerned.

Why It Wasn't So Smart

Even though RayStar's development teams were working on very different services—which didn't ***seem*** to allow them room to work together, by refusing to acknowledge each other, or even speak, they were being highly inefficient in their management processes—wasting RayStar's resources by replicating processes—and putting the company at risk of violating regulations and policies crucial to the company's (financial and legal) well-being by not seeking means to employ more comprehensive governance.

A Better Approach

The bottom line was that RayStar—with its development teams spread out and disconnected as they were (and chose to be)—was crying out for governance. What is often unrecognized is that perhaps the most important aspect of governance is the human element. Much of what has been discussed in this paper centers on such things as "communication", "partnerships" and "business buy-in."

So it should come as no surprise then that the bulk of the recommendations for RayStar consist of talking ... and then setting up processes. First comes talking. This can be truly difficult—especially in organizations in which there is a great deal of political resistance associated with coming together. We might imagine this to be the case with the individual "kings" at RayStar and their hills. But there is usually a big payoff for the first two parties who finally start talking. More often than not, they are the ones who get to set the standards! (Tip: talking first can be the secret to holding onto your kingdom ...)

Once the discussion begins, it will consist over time of identifying all the existing processes, and inserting governance checkpoints and notification points (e.g., "Notify such-and-such that *Service A* has been created"). A standing group ought to also be incorporated to make certain that all processes are complied with. All these things said, while governance at its core is about communication, it wouldn't hurt to deploy a SOA governance solution to simplify and automate these processes and to make absolutely certain that identified policies are complied with—particularly as these policies change over time.

And, most importantly, besides pairing up the IT heads from the various divisions, IT ***must*** pair up with business partners. To be successful over time, IT ought to choose business partners with immediate needs, who are vocal, and who can quantify the value new services can bring them. With such partnerships in hand, IT will have strong cases for moving forward in ways that will make sense to the entire organization—right up to the Board of Directors.

PARTING THOUGHTS

Building an SOA can be challenging, even agonizing. Sometimes ideas that look good on paper are not so good in practice. Many actually turn out to be worst practices. But, as this guide illustrates, you can learn a lot from others' mistakes.

For the unabridged truth, you can download the complete SOA Worst Practices whitepapers, volumes one and two, from the Progress Software web site. In these papers, we examine:

- Why reuse, not the number of Web services, is the true indicator of SOA value
- Why standard SOAP stacks are key to performance
- When to build an SOA and the business benefits
- How to build an SOA by adding a wrapper to existing applications
- Why an XML firewall an SOA does not make
- Why it's important to understand the purpose and true benefits of an SOA before moving forward
- What key business rules an SOA strategy should address
- Why SOA does not require a "rip and replace" approach
- Why cloning and reconfiguring applications is not the way to go
- Why you should be careful about sharing your WSDL
- Why your partners can be your best allies when it comes to securing your schema validation
- Why governance is not just a check-once, design-time concern
- Why your organization is on the hook to comply with all relevant regulations and policies—at all times
- Why IT teams must take an integrated approach to planning for security, management and governance policies and processes
- How you can get around building dedicated infrastructure capacity to serve your most highly valued customers
- How to keep track of unexpected consumers of the Web services which are now built into today's enterprise packaged applications
- How to avoid getting lulled into complaisance by the capabilities of tools that are only part of the total governance equation
- Why it's important for IT and business stakeholders to be in alignment from the start about the goals of the SOA and individual services
- Why it's critical for governance to be institutionalized throughout the development cycle—and particularly across organizational "silos" in large firms.

More important, you can gain the wisdom needed to make better decisions as you embark on an SOA initiative.

ABOUT PROGRESS SOFTWARE CORPORATION

Progress Software Corporation (Nasdaq: PRGS) provides application infrastructure software for the development, deployment, integration and management of business applications. Our goal is to maximize the benefits of information technology while minimizing its complexity and total cost of ownership. Progress can be reached at www.progress.com or +1-781-280-4000.

Progress® Actional® Web services management and SOA governance products provide visibility, security and control of the activities of services and end-to-end business processes in a runtime environment. Actional Web services management and SOA governance capabilities include system and process-level visibility, as well as policy enforcement across an SOA infrastructure deployed on any combination of platforms.

The Progress DataXtend™ product family provides data integration for distributed applications, delivering real-time views of shared data, in the form that applications need. DataXtend offers a unique approach to data management problems, employing a common semantic data model to create sophisticated data transformations, enabling organizations to integrate heterogeneous data sources with no disruption to existing applications.

Sonic™ products from Progress Software help IT organizations achieve broad-scale interoperability of IT systems and the flexibility to adapt these systems to rapidly changing business needs. The Sonic products include SonicMQ®, the industry's only continuously available JMS enterprise messaging system, and Sonic ESB®, the world's first and market-share leading ESB. Sonic products simplify the integration and flexible reuse of diverse and often proprietary business systems by manipulating them as modular, standards-based services which can be rapidly combined to serve the business in new ways.

www.progress.com

Worldwide and North American Headquarters

Progress Software, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000

UK and Northern Ireland

Progress Software, 210 Bath Road, Slough, Berkshire, SL1 3XE England Tel: +44 1753 216 300

Central Europe

Progress Software, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49 6171 981 127