SOA and API Convergence Strategy and Tactics
Table of Contents

Introduction.................................................................................................................................................................................................. 03
SOA and API Business Goals.......................................................................................................................................................... 03
SOA Business Perspective and API Economy Echo.......................................................................................................................... 04
SOA Technical Perspective and API Specialization...................................................................................................................... 04
Schism Between API and SOA, and Pragmatic Reconciliation.................................................................................................. 04
Pragmatic REST API Focus........................................................................................................................................................... 05
Pragmatic SOA Focus.................................................................................................................................................................... 05
Pragmatic Reconciliation............................................................................................................................................................... 06
Defining an Appropriate SOA and API Mindset................................................................................................................................ 06
Big SOA Mindset............................................................................................................................................................................. 07
Small SOA Mindset.......................................................................................................................................................................... 08
API-Access Mindset......................................................................................................................................................................... 08
API-Centric Enterprise Mindset....................................................................................................................................................... 09
Build a Services and APIs.............................................................................................................................................................. 09
When to Create a Service?.............................................................................................................................................................. 09
When to Create RESTful APIs......................................................................................................................................................... 11
How to Approach Service, API, and Application Governance.................................................................................................. 11
SOA Governance............................................................................................................................................................................... 12
Service Portfolio is [A] in SOA......................................................................................................................................................... 12
API Governance............................................................................................................................................................................... 13
Converging API Governance and SOA Governance.................................................................................................................. 13
Common Description Needs........................................................................................................................................................... 14
Governance Balance......................................................................................................................................................................... 15
Conclusion....................................................................................................................................................................................... 15
Recommended Resources............................................................................................................................................................... 17
Introduction

While everyone acknowledges API and Service Oriented Architecture (SOA) are best practice approaches to solution and platform development, the learning curve and adoption curve can be steep. To gain significant business benefits, teams must understand their IT business goals, define an appropriate SOA & API mindset, describe how to implement shared services and popular APIs, and tune governance practices.

SOA is a design discipline that focuses on implementing shared functionality as reusable services. A core SOA principle is to maintain a clean separation of concerns between services implementing shared functionality and applications that consume them. This design approach enables any type of application to consume the service. It hides implementation details from the consuming applications, and enables more flexibility and adaptability in the systems.

The core SOA design unit is a shared, reusable service that implements a discrete piece of functionality. The service exposes its functionality through a well-defined interface. Any application that requires the functionality uses the service. A single service may be used across multiple applications.

APIs are a strategic component supporting your SOA initiative.

When development teams publish APIs, they explicitly separate service interface from service implementation. Managed API endpoints are lightweight proxies enforcing security, monitoring usage, and shaping traffic. The API proxy enables a clear separation of concern between consumer interface contract and back-end service implementation.

A managed API is:

- Actively advertised and subscribe-able
- Available with an associated, published service-level agreement (SLA)
- Secured, authenticated, authorized and protected
- Monitored and monetized with analytics

While services should exhibit managed API characteristics, most service endpoints are deployed on platforms that don’t deliver management characteristics. By applying the API façade pattern, teams can layer network addressable endpoints, monitor usage, enforce usage limits, shape traffic, and authorize consumers.

SOA and API Business Goals

During the SOA craze days in the past, proponents pitched SOA’s lofty benefits from both business and technical perspectives. The benefits are real, yet sometimes very difficult to obtain. Surprisingly, today’s API proponents target similar benefits, but with an execution twist.
SOA Business Perspective and API Economy Echo

SOA can be a strategy to align IT assets with business capabilities, business resources, and business processes. SOA’s strong focus on sharing and re-use can optimize IT asset utilization. Most intriguingly, SOA was promised to re-invent business-to-business interactions, enable better partner relationships, and support process networks\(^1\). External services were seen as a mechanism to extend an enterprise’s economic reach by reducing interaction costs, incorporating external business capabilities, enabling business specialization, and creating higher-value solutions that extend business processes across a partner network.

The current API economy buzz co-opts the SOA business value proposition, injects lessons learns, and rides popular industry trends (i.e. REST, Internet of Everything, mobile, Cloud services).

SOA Technical Perspective and API Specialization

From a technical perspective, an SOA must exhibit three key design principles; service orientation, clean separation of concerns, and loose coupling. Service orientation is gauged by service reusability, granularity, and architecture simplification. Clean separation of concerns is gauged by testing separation of business logic from infrastructure, interface from implementation, and interface from capability. Loose coupling is gauged by measuring interoperability, transaction control, and mediated interactions.

On the surface, RESTful APIs are simply a specialized version of web services, and provide similar technical benefits. Both REST API design and SOA service design intend to expose discrete functionality via a well-defined interface. The API endpoint and the service endpoint both serve as a core design unit, and teams deliver technical solutions by accessing, aggregating, composing, and orchestrating endpoint interactions. Successful and scalable API and service-based solutions both require Internet messaging infrastructure, service level management, and security.

Schism Between API and SOA, and Pragmatic Reconciliation

While both API and SOA proponents have similar business and technical goals, a large execution schism exists between the two camps. The schism between pragmatic REST API and pragmatic SOA is caused by differences in strategic focus.

Teams ‘doing REST’ and ‘building APIs’ commonly focus on overcoming technical and business adoption barriers by pursuing incremental build-outs and demonstrating concrete, core-business use cases without introducing complex technology. SOA teams commonly focus on obtaining efficiencies at scale, achieving enterprise standardization, centralizing decisions, and satisfying complex non-functional requirements.

---

\(^1\) The Only Sustainable Edge, John Hagel III & John Seely Brown, 2005
http://www.johnseelybrown.com/readingTOSE.pdf
Pragmatic REST API Focus

REST is an architectural style of system development imposing a series of constraints on service interactions. Taken together, the constraints allow beneficial properties to emerge, namely simplicity, scalability, modifiability, reliability, visibility, performance, and portability. Systems that satisfy these constraints are RESTful. A RESTful design approach can foster many benefits:

- Make data and services maximally accessible
  - Low barrier to entry
  - Extend reach towards the largest possible audience
  - Make API/service consumable by the largest number of user agents
- Make data and services evolvable
  - Extend the system at runtime
  - Alter resources without impacting clients
  - Direct client behavior dynamically
- Make systems scalable, reliable, and high performing
  - Simple
  - Cacheable
  - Atomic

While RESTful design benefits support SOA goals, the strategic focus of Pragmatic REST differs from many SOA initiatives. Pragmatic REST API design teams focus on bottoms-up adoption scenarios, approachable protocols/formats (e.g. HTTP, JSON, DNS), permissive interface definitions, and simpler interaction models (i.e. retry over guaranteed delivery).

Pragmatic SOA Focus

Pragmatic SOA focuses on service-oriented patterns that increase software asset value. The fundamental service-oriented patterns are:

- Share and re-use assets
- Consolidate redundant functionality into fewer moving parts
- Conform projects to common standards and best practices

Applying these three patterns will reduce complexity within an IT environment and lead to greater agility, i.e. the ability to build applications faster and modify them quickly to address changing requirements. The service-oriented patterns force development teams to evaluate how software asset capabilities meet the needs of business stakeholders.

Pragmatic SOA teams don’t force common (yet complicated) standards. Pragmatic SOA teams offer useful business capabilities, reduce adoption friction, and deliver exceptional service values.

Pragmatic SOA teams don’t preach difficult best practices. Pragmatic SOA teams simplify best practice adoption by mentoring teams and delivering automated governance that makes the right thing to do the easy team to do.
Pragmatic SOA teams are mindful of skill gaps and adoption hurdles. Pragmatic teams offer accelerator packs (i.e. infrastructure, tooling, frameworks, and API/service building blocks) that reduce training, increase self-service adoption, and accelerate project delivery.

Pragmatic SOA teams balance enterprise governance with project autonomy. Instead of erecting development and registration barriers, successful teams foster service development, service sharing, and service adoption by introducing mechanisms to promote services, mediate interactions, harden service levels, and facilitate self-service adoption. You may recognize these mechanisms as being the core of API management.

**Pragmatic Reconciliation**

REST is different from—although not incompatible with—SOA. Services can be RESTful, and RESTful resources can be services. Like SOA, REST is an architectural discipline defined by a set of design principles, and REST also imposes a set of architectural constraints. REST uses a resource-centric model, which is the inverse of an object-centric model (i.e. behaviors are encapsulated by resources). In REST, every “thing” of interest is a resource. When modeling a RESTful service (aka APIs), the service’s capabilities are encapsulated and exposed as a set of resources.

Because SOA presents an architectural goal state at odds with a long-lived legacy IT portfolio, SOA is a long-term architectural journey, and not a short-term implementation initiative. Because APIs interconnect business capabilities inside and outside the organization, APIs can provide a platform for business stakeholders sponsoring enterprise IT renewal and pragmatic business execution.

**Defining an Appropriate SOA and API Mindset**

SOA represents a significant paradigm shift in application development techniques. SOA is a software design discipline in which application and infrastructure functionality are implemented as shared, re-usable services. Each service implements a discrete task, and any application that needs to perform the task uses the shared service to do so. Teams create applications by assembling the appropriate services. After teams implement business functionality as services, an organization, their partners, and their customers should be able to mix and match these services and rapidly create new applications to support changing business requirements. The API economy and API business to developer messages reinforce the traditional SOA value proposition. The benefits of using the WSO2 platform for implementing these services include:

Because SOA presents an architectural goal state at odds with a long-lived legacy IT portfolio, SOA is a long-term architectural journey, and not a short-term implementation initiative. Because APIs interconnect business capabilities inside and outside the organization, APIs can provide a platform for business stakeholders sponsoring enterprise IT renewal, pragmatic business execution, and quick wins.

The last ten years of SOA promotion, implementation, and evaluation have cultivated two distinct ways to approach SOA; Big SOA Mindset and Small SOA Mindset. Recent API strategies have centered on pursuing either an API-Access Mindset or API-centric Enterprise Mindset.
Big SOA Mindset

Big SOA requires organizations to scale adoption across multiple consumers by establishing a shared understanding of business and technical domains. Each service is part of a bigger picture, and a well-designed service plugs into existing business processes and supports defined business requirements. Effective service adoption within an enterprise requires significant planning, coordination, and governance. Organizations must be cautious not to let a Big SOA initiative fail due to analysis paralysis or inter-enterprise feudal fiefdoms.

Big SOA initiatives focus on enterprise governance processes, enterprise-wide re-use, and portfolio consolidation. Achieving these big goals requires structurally changing design-time processes and overcoming significant cultural bias that lead to inhibiting design, accounting, control, and operational ramifications. Traditional Big SOA initiatives can succeed, but only if top-level organizational support (e.g. C-level) overcomes organizational inertia and project teams bridge silos and operate as one team.

When tackling Big SOA, teams need to find an effective entry point. Some teams start with an IT-driven initiative or slipstream on important business-driven transformation initiatives.

An IT-driven Big SOA approach can be very effective when focusing on building shared infrastructure services, such as security, identity, auditing, monitoring, and cloud management. These efforts can deliver rapid returns on investment, and they don’t require extensive collaboration with the business units for success. On the other hand, this approach delays engaging the business and may provide minimal return on investment (ROI) recognition unless incorporated into more visible risk management, cyber-security, or customer experience initiatives.

A business-driven Big SOA approach is often triggered by a top-down imperative to redesign business processes and/or the customer experience. By design, these business transformation efforts must be highly coordinated across IT and the various business units. When considering pursuing Big SOA, slipstream service portfolio investment behind funded business projects.

A Big SOA initiative is not a short quarterly project, but a multi-year program containing several roadmap steps. To maximize and accelerate success, the team embarking on Big SOA should incorporate a structured transformation program into their journey:

- Set up a cross-functional SOA Working Group
- Develop a SOA Adoption Plan
- Define Target Service Portfolio
- Develop a Business Case
- Plan and Fund Development of SOA Infrastructure
- Establish New Roles
- Plan Training and Mentoring for Staff
- Develop Corporate Policies, Guidelines, and Best Practices
- Institute SOA Governance Processes
- Establish New Incentives that Reward Good Behavior
- Identify Candidate Projects
- Establish Priorities
- Re-assess Your Software Development LifeCycle (SDLC)
Many teams embarking on SOA do not have the authority, span of control, or influence to implement required Big SOA activities. With Big SOA success un-obtainable in many organizations, team members who want to ‘do SOA’ and demonstrate business value often focus on small SOA.

**Small SOA Mindset**

A Small SOA approach implements SOA principles on a project-by-project basis. This approach incurs less risk, but produces a smaller return. Typically there’s limited coordination across projects, and Small SOA doesn’t require as much cultural, arm-twisting, or political unrest. Using a Small SOA process, the organization can slowly build a portfolio of services, but with limited coordination, the services are less likely to be re-usable or shared across multiple consumers.

Small SOA initiatives often focus on run-time environment concerns instead of design-time concerns. Popular run-time environment concerns include enabling flexible communication styles, interaction patterns, and mediation mechanisms that facilitate integration and promote loose coupling. Example communication styles include synchronous, asynchronous, one-way, and request response messaging. Commonly supported interaction patterns include peer-to-peer, brokered delivery, hub-and-spoke, publish/subscribe, and orchestrated workflow, which are used to bridge the gap between consumer-provider availability, reliability, and topology.

Mediation mechanisms reduce the need to provide symmetric messaging platforms where both consumer and provider communicate using the same protocol, message format, and communication style. Mediation mechanisms also encapsulate implementation details related to location, versioning, and identity domain.

Adopting an IT-driven Small SOA approach requires a cautious approach to not let the effort focus exclusively on technology rather than design, culture, and IT goals. Successful IT-driven Small SOA teams promote consumer adoption stories, track service subscribers, and publicize usage growth.

A business-driven Small SOA approach can result in compelling singular success stories, which will encourage SOA adoption. However, business asset re-use is doubtful without cross-team collaboration and promotion. Uncoordinated and ungoverned project-by-project efforts can generate a chaotic situation resulting in little or no reuse. How many of your services are shared, and how many services simply facilitate point-to-point integration?

**API-Access Mindset**

Similar to business-driven Small SOA, an API-access approach results in compelling success stories, which encourages further adoption. RESTful APIs provide an approachable technology platform for teams integrating IT and business capabilities with mobile applications, browser-based JavaScript applications, or machine-to-machine (M2M) shell scripts.

While APIs can advance a SOA strategy, APIs do not have to follow SOA principles (i.e. loose coupling, separation of concerns, service-orientation), and API teams predominantly focus on integration concerns that reinforce capability silos. API project teams bridge span of control domains and integrate external APIs into project solutions.
API-Centric Enterprise Mindset

Rather than focus on IT sharing and re-usability concerns (the SOA focus), API proponents evangelize how teams can redesign their business, increase revenue growth, and retain customers by building an API-centric Enterprise.

API-centric Enterprises move beyond destination ecommerce to embrace decentralization, personalization, contextualization, gamification, and dynamic distribution channels. Technology trends embracing these concepts include mobile, machine-to-machine, person-to-person, and business-to-developer channels. In each case, APIs are the connecting glue across distributed solution actors and components.

API-centric Enterprise teams use APIs to swiftly assemble solutions using external and internal building blocks. Because APIs extend reachability, facilitate consumability, and reduce technical barriers, power business users and developers can rapidly snap together solutions. Common API building blocks include customer, order, product, logistics, map, email, SMS, weather, or traffic.

In addition to gaining internal productivity benefits, API-centric Enterprise organizations reach more customers and generate greater business activity. APIs facilitate interconnecting business processes across the extended value chain. Partners, suppliers, distributors, and customers can readily tap into a business capability offered as an API, and increase business interaction over the API channel.

Build a services and APIs

Instead of creating consistent service architecture and demonstrating service re-use, teams often inadvertently produce Just a Bunch of Web Services (JBOWS) or Just a Bunch of REST Services (JBORS). A single application often consumes a service, and a spaghetti web of One-to-One connections exists between service provider endpoints and consumers. Many teams find an SOA or REST focus may not improve IT agility, but result in simply swapping out IT toolsets, message formats, and protocols.

IT infrastructure, design processes, governance, culture, and incentives should support reaching SOA and API business goals by encouraging asset sharing and re-use, consolidating functionality, and driving development team adoption.

To create a better architecture environment and gain API-centric agility, teams must understand:

- When to create services
- When to create APIs
- How to approach service and API governance
- How services and APIs impact application governance

When to Create a Service?

Create a service when sharing business capability or business data across network or run-time application boundaries.
A service should implement a reasonably autonomous business task and not be designed as a fine-grained component with interdependencies on other components. Developers and business analysts are more likely to understand the purpose of the service when the service exposes a discrete task within a business process. Designing services for business task granularity reduces the complexity of the service interactions and simplifies application maintenance. Examples of business tasks that are compatible with a service-oriented approach include “submit order,” “retrieve customer record,” and “pay bill.”

The service should expose the capability through multiple styles of interfaces (e.g. HTTP/JSON, XML/JMS, SOAP/XML, asynchronous, synchronous), and RESTful APIs are just an interface style. Ideally, the interface style is just an implementation detail, and important management characteristics (i.e. security, service level enforcement, usage tracking) are available across all interface style channels (e.g. resource oriented, publish/subscribe, method invocation). Figure 1 illustrates façade pattern implementation:

Separating external contract and external protocols from internal representation positively impacts the ability to evolve the service implementation over time. When a clean separation of interface from implementation exists, a developer can modify the implementation without impacting the calling applications that use the service.

However, decoupling consumer and provider from a shared application platform, and decoupling interface from implementation raises additional concerns. For example, difficulties arise when attempting to seamlessly flowing operational semantics (i.e. identity, service levels, authorization) across dissimilar platforms and a distributed environment. Teams rely on middleware infrastructure to bridge the semantics across all participating parties and components.

Because separating interface from implementation introduces complexity and extra development effort, many teams tightly bind interface to implementation. By following architectural best practices and introducing API facades or proxy endpoints (with development automation), teams increase solution maintainability and adaptability.
When to Create RESTful APIs

Create a RESTful API when sharing a service outside a domain of control (i.e. business unit, organization), targeting the widest possible reach and consumption, offering the service across native web infrastructure, or interested in maximizing asymmetric evolution between service clients, interface, and implementation.

When development teams publish API facades in front of existing services, they explicitly separate service interface from service implementation. API endpoints are lightweight proxies enforcing security, monitoring usage, and shaping traffic. The proxy enables a clear separation of concern between consumer interface contract and back-end service implementation.

While application servers, enterprise service bus nodes, or data service hosts could publish API endpoints, API gateways are the preferred managed API delivery mechanism. A managed API is:

- Actively advertised and subscribe-able
- Available with an associated, published SLA
- Secured, authenticated, authorized, and protected
- Monitored and monetized with analytics

RESTful APIs expose a resource-oriented interface. The interface is a set of resources, each exposing a uniform API. To create a RESTful API:

1. Give everything an ‘ID’
2. Link things to each other
3. Use standard HTTP methods
4. Allow for multiple message format representations
5. Reduce shared state

How to Approach Service, API, and Application Governance

SOA initiative success requires creating loosely coupled consumer-provider connections, enforcing a separation of concerns between consumer and provider, exposing a set of re-usable, shared services, and gaining service consumer adoption. Many development teams publish services, yet struggle to create a service architecture that is widely shared, re-used, and adopted across internal development teams. Instead of creating consistent service architecture and demonstrating service re-use, teams inadvertently produce Just a Bunch of Web Services (JBOWS) or Just a Bunch of REST Services (JBORS). A single application often consumes a service, and a spaghetti web of One-to-One connections exists between service provider endpoints and consumers. Many teams find a SOA or REST focus may not improve IT agility, but result in simply swapping out IT toolsets, message formats, and protocols.

SOA governance, API governance, and application governance can stand in the gap and improve architectural coherence.
SOA Governance

In many organizations, SOA initiatives will deliver point-to-point integration instead of architecture unless existing governance programs are modified to mitigate risk factors and support SOA principles. Delaying service governance often results in creation of non-reusable services, proliferation of multiple business domain definitions, and increasing portfolio maintenance cost.

An effective SOA governance program controls the development and operation of service oriented systems, and is implemented using policies, processes, metrics, and organization:

- Policies specify the “right” way to do things. They codify laws, regulations, corporate guidelines, and best practices.
- Processes are activities that provide an opportunity to test a project or artifact for compliance with policies, and to make a go/no-go decision. Some processes are automatic and system driven; others require human effort.
- Metrics provide visibility into the governance program, and are required to measure compliance and verify policy enforcement.
- The organization should encourage a culture that supports and rewards good governance practices.

An SOA governance program should provide guidance for the entire service lifecycle, including creation, testing, provisioning, utilization, management, and versioning.

SOA governance infrastructure components provide tools and services that support the governance program. They provide mechanisms to manage and maintain governance policies; they provide mechanisms to impose checkpoints during various phases of the software development lifecycle (SDLC) and verify that services, APIs and/or projects comply with these policies; they also provide mechanisms that support manual and automatic approval and exception processes; and they enable integration with traditional SDLC tools and information technology (IT) management and governance systems.

Some SOA governance infrastructure components address development-time governance, and other components address runtime governance. The service registry component bridges the development and runtime components.

Service Portfolio is [A] in SOA

A key SOA end-goal is a service portfolio exhibiting a clean architecture. Technical and industry vertical business domain models are useful starting points for development of an organization’s unique service portfolio. Service design and interface design are small pieces in a larger set of programs that must be managed in order to establish highly effective IT service delivery, collaboration, and trust.

Initiating complementary projects such as an application rationalization, business process management or enterprise architecture is an excellent mechanism to understand IT dynamics, decompose the business domain, and determine what software assets (and services!) should support business processes.
API Governance

API governance is heavily influenced by IT business goals and objectives. Leading API governance platforms provide analytics supporting the assessment of IT business value. The platform should capture service tier subscription information, collects usage statistics, present productivity metrics, and integrate with billing and payment systems.

API governance encompasses API subscriptions and API promotion meta-data. Governance activities managing API promotion meta-data include rationalizing keyword tags used to categorize APIs, and developer documentation content management. The governance process should enforce design-time checkpoints before API publication.

API Economy metrics include adoption and usage. API governance establishes API adoption (by version, by API) and usage (by version, by API) processes, policies, and metrics. By understanding API adoption and usage, API business owners and API architects can intelligently invest future development resources, properly plan API infrastructure scale, and rationalize the API portfolio.

Converging API Governance and SOA Governance

To efficiently and effectively deliver IT solutions, teams must synchronize API governance activities and service governance activities. API management provides a straightforward set of lifecycle stages (e.g., created, published, deprecated, retired, blocked) that may be customized by the development team. Governance registries facilitate service metadata management and governance in design, implementation, test, and run-time operations.

Figure 2 below depicts the intersection of the two governance views.
Common Description Needs

When developers consume, aggregate, orchestrate, or compose APIs or services across organizational boundaries, documentation and contracts describing how to interact with the API or service strongly influence adoption. API proponents eschew heavyweight, formal service interface contracts for human readable documentation. The RESTafarian backlash against schema, description languages, and machine-readable policy is starting to abate.

REST best practices are starting to influence design and description. Whether building RESTful APIs or SOA services, teams should:

- Not publish endpoints exposing rigid data models and requiring complicated interaction patterns
- Document how session context and session state influences interactions
- Document the message model
- Describe usage limits and security concerns
  (i.e. acceptable authentication credentials, authorization protocols)
- Hide internal details from the external interface

SOA and API tactics should include defining metadata and policy frameworks that provide the scaffolding required to identify, classify, and describe consumption candidates. Metadata is the design and development time information used to describe a service or API. Policies are used to map metadata to deployed service instances and endpoints. Metadata can span the following concerns:

- Overview (e.g. name, description)
- Life Cycle Attributes (e.g. version, version relationships, lifecycle status)
- Classification (e.g. basic, composite, infrastructure, business)
- Endpoint Deployment Attributes (e.g. protocols, location, WS-* specifications)
- Data Model (e.g. JSON Schema, RAML, XML Schema, WSDL, version, semantics, validation)
- Service Level Requirements and Policies (e.g. availability, capacity, responsiveness, security, transaction rate)
- Mediation (e.g. routing, queuing, caching, transformation)
- Dependency Attributes (e.g. APIs, services, databases, directories, frameworks)
- Physical Instance Dependencies (e.g. application platform, security, management)
- Business Process Model (e.g. UML diagram, business classification)
- Contract information (e.g. consumers, providers, utilization)
- Usage Guidelines (e.g. time of day, availability, # of users. throughput)
- Accounting or remuneration options (e.g. pay per use, subscription, chargeback amount)

Metadata enables formal description of the endpoints and promotes discovery and re-use by other teams. Most organizations don’t start with a full inventory effort, but instead simplify the project’s scope. They work towards describing a single candidate service/API that is core to the business, refining their metadata taxonomy, and gaining buy-in from solution teams on possible re-use.
Governance Balance

Achieving a productive balance between helpful guidance and stifling control requires understanding impediments and team culture. SOA/API initiatives often stall due to:

- Insufficient education, training, and mentoring
- Mistrust or lack of collaboration across organizations
- Conflicting incentives
- Poor service/API design
- Lack of metrics
- Lack of governance

Guidance is necessary, but if governance processes are onerous, they will cause resentment and resistance. Whenever possible, processes should be seamless and automatic. For example, compliance checking can be performed automatically during code check-in or during service registration. Many policy management systems provide tools for automatic compliance checking. Look for products that seamlessly integrate with developer tooling to make it as simple as possible for developers to identify and fix compliance issues.

Contract management systems can be invaluable in supporting governance of the consumer provisioning process, as well as service lifecycle management, enhancement requests, and service versioning.

Successful Big SOA, Small SOA, API-access, and API-centric Enterprise success is predicated on pervasive adoption, sharing, and re-use across project silos, organizational domains, and development communities. Successful governance initiatives implement the following governance activities:

- Establish design review board
- Tighten control of local class and information models
- Adopt shared funding model
- Implement compliance monitoring
- Create common service level agreement definitions
- Create an inventory of application assets and integration connections
- Implement governance review processes to manage change in the inventory
- Extend software development life cycle governance to embrace service-orientation

Conclusion

SOA can be a strategy to align IT assets with business capabilities, business resources, and business processes. SOA’s strong focus on sharing and re-use can optimize IT asset utilization. Most intriguingly, SOA was promised to re-invent business-to-business interactions, enable better partner relationships, and support process networks. External services were seen as a mechanism to extend an enterprise’s economic reach by reducing interaction costs, incorporating external business capabilities, enabling business specialization, and creating higher-value solutions that extend business processes across a partner network.

The current API economy buzz co-opts the SOA business value proposition, injects lessons learns, and rides popular industry trends (i.e. REST, Internet of Everything, mobile, Cloud services).
APIs are a strategic component supporting your SOA initiative. On the surface, RESTful APIs are simply a specialized version of web services, and provide similar technical benefits. Both REST API design and SOA service design intend to expose discrete functionality via a well-defined interface. Successful and scalable API and service-based solutions both require managed endpoints.

REST is an architectural style of system development imposing a series of constraints on service interactions. Taken together, the constraints allow beneficial properties to emerge, namely simplicity, scalability, modifiability, reliability, visibility, performance, and portability. Systems that satisfy these constraints are RESTful.

While RESTful design benefits support SOA goals, the strategic focus of Pragmatic REST differs from many SOA initiatives. Pragmatic REST API design teams focus on bottoms-up adoption scenarios, approachable protocols/formats (e.g. HTTP, JSON, DNS), permissive interface definitions, and simpler interaction models (i.e. retry over guaranteed delivery).

Teams ‘doing REST’ and ‘building APIs’ commonly focus on overcoming technical and business adoption barriers by pursuing incremental build-outs and demonstrating concrete, core-business use cases without introducing complex technology. SOA teams commonly focus on obtaining efficiencies at scale, achieving enterprise standardization, centralizing decisions, and satisfying complex non-functional requirements.

Pragmatic SOA and Pragmatic API approaches have significant overlap and commonality. Both Pragmatic SOA teams and Pragmatic REST API teams don’t force common (yet complicated) standards, don’t preach difficult best practices, balance enterprise governance with project autonomy, and are mindful of skill gaps and adoption hurdles. A successful SOA adoption strategy requires fundamental changes to application design and development techniques and reducing API/service adoption friction. SOA governance, API governance, and application governance must intersect and coexist for an organization to move towards a rational architectural goal state.

Because SOA presents an architectural goal state at odds with a long-lived legacy IT portfolio, SOA is a long-term architectural journey, and not a short-term implementation initiative. Because APIs interconnect business capabilities inside and outside the organization, APIs can provide a platform for business stakeholders sponsoring enterprise IT renewal and pragmatic business execution.

Similar to business-driven Small SOA, an API-access approach results in compelling success stories, which encourages further adoption. RESTful APIs provide an approachable technology platform for teams integrating IT and business capabilities with mobile applications, browser-based JavaScript applications, or machine-to-machine (M2M) shell scripts.

API-centric Enterprise organizations reach more customers and generate greater business activity. APIs facilitate interconnecting business processes across the extended value chain. Partners, suppliers, distributors, and customers can readily tap into a business capability offered as an API, and increase business interaction over the API channel. API-centric Enterprises move beyond destination ecommerce to embrace decentralization, personalization, contextualization, gamification, and dynamic distribution channels. Technology trends embracing these concepts include mobile, machine-to-machine, person-to-person, and business-to-developer channels. In each case, APIs are the connecting glue across distributed solution actors and components.
Recommended Resources

- Promoting service reuse within your enterprise and maximizing SOA success
- Application Services Governance: Automate IT Best Practices and Enforce Effective and Safe Application Service Delivery
- Implementing API Facade using WSO2 API Management Platform
- Enterprise Integration and ESB

About WSO2

WSO2 is the only company that provides a completely integrated enterprise application platform for enabling a business to build and connect APIs, applications, Web services, iPaaS, PaaS, software as a service and legacy connections without having to write code; using big data and mobile; and fostering reuse through a social enterprise store. Only with WSO2 can enterprises use a family of governed secure solutions built on the same code base to extend their ecosystems across the cloud and on mobile devices to employees, customers and partners in anyway they like. Hundreds of leading enterprise customers across every sector—health, financial, retail, logistics, manufacturing, travel, technology, telecom and more—in every region of the world rely on WSO2’s award-winning, 100% open source platform for their mission-critical applications. To learn more, visit http://wso2.com or check out the WSO2 community on the WSO2 Blog, Twitter, LinkedIn, Facebook, and FriendFeed.

Check out more WSO2 Whitepapers and WSO2 Case Studies.

For more information about WSO2 products and services, please visit http://wso2.com or email bizdev@wso2.com