

## **The Annotated SOA Manifesto**

Commentary and Insights about the SOA Manifesto from Thomas Erl

**Service orientation is a paradigm that frames what you do. Service-oriented architecture (SOA) is a type of architecture that results from applying service orientation.**

From the beginning it was understood that this was to be a manifesto about two distinct yet closely related topics: the service-oriented architectural model and service orientation, the paradigm through which the architecture is defined. The format of this manifesto was modeled after the Agile Manifesto, which limits content to concise statements that express ambitions, values, and guiding principles for realizing those ambitions and values. Such a manifesto is not a specification, a reference model or even a white paper, and without an option to provide actual definitions, we decided to add this preamble in order to clarify how and why these terms are referenced in other parts of the manifesto document.

**We have been applying service orientation...**

The service orientation paradigm is best viewed as a method or an approach for realizing a specific target state that is further defined by a set of strategic goals and benefits. When we apply service orientation, we shape software programs and technology architecture in support of realizing this target state. This is what qualifies technology architecture as being service-oriented.

**...to help organizations consistently deliver sustainable business value, with increased agility and cost effectiveness...**

This continuation of the preamble highlights some of the most prominent and commonly expected strategic benefits of service-oriented computing. Understanding these benefits helps shed some light on the aforementioned target state we intend to realize as a result of applying service orientation.

Agility at a business level is comparable to an organization's responsiveness. The more easily and effectively an organization can respond to business change, the more efficient and successful it will be to adapting to the impacts of the change (and further leverage whatever benefits the change may bring about).

Service orientation positions services as IT assets that are expected to provide repeated value over time that far exceeds the initial investment required for their delivery. Cost-effectiveness relates primarily to this expected return on investment. In many ways, an increase in cost-effectiveness goes hand-in-hand with an increase in agility; if there is more opportunity to reuse existing services then there is generally less expense required to build new solutions.

"Sustainable" business value refers to the long-term goals of service orientation to establish software programs as services with the inherent flexibility to be continually

composed into new solution configurations and evolved to accommodate ever-changing business requirements.

**...in line with changing business needs.**

These last six words of the preamble are key to understanding the underlying philosophy of service-oriented computing. The need to accommodate business change on an on-going basis is foundational to service orientation and considered a fundamental over-arching strategic goal.

**Through our work we have come to prioritize:**

The upcoming statements establish a core set of values, each of which is expressed as a prioritization over something that is also considered of value. The intent of this value system is to address the hard choices that need to be made on a regular basis in order for the strategic goals and benefits of service-oriented computing to be consistently realized.

**Business value over technical strategy**

As stated previously, the need to accommodate business change is an overarching strategic goal. Therefore, the foundational quality of service-oriented architecture and of any software programs, solutions, and eco-systems that result from the adoption of service orientation is that they are business-driven. It is not about technology determining the direction of the business, it is about the business vision dictating the utilization of technology.

This priority can have a profound ripple effect within the regions of an IT enterprise. It introduces changes to just about all parts of IT delivery lifecycles, from how we plan for and fund automation solutions, to how we build and govern them. All other values and principles in the manifesto, in one way or another, support the realization of this value.

**Strategic goals over project-specific benefits**

Historically, many IT projects focused solely on building applications designed specifically to automate business process requirements that were current at that time. This fulfilled immediate (tactical) needs, but as more of these single-purpose applications were delivered, it resulted in an IT enterprise filled with islands of logic and data referred to as application “silos.” As new business requirements would emerge, either new silos were created or integration channels between silos were established. As yet more business change arose, integration channels had to be augmented, even more silos had to be created, and soon the IT enterprise landscape became convoluted and increasingly burdensome, expensive, and slow to evolve.

In many ways, service orientation emerged in response to these problems. It is a paradigm that provides an alternative to project-specific, silo-based, and integrated

application development by adamantly prioritizing the attainment of long-term, strategic business goals. The target state advocated by service orientation does not have traditional application silos. And even when legacy resources and application silos exist in environments where service orientation is adopted, the target state is one where they are harmonized to whatever extent feasible.

### **Intrinsic interoperability over custom integration**

For software programs to share data they need to be interoperable. If software programs are not designed to be compatible, they will likely not be interoperable. To enable interoperability between incompatible software programs requires that they be integrated. Integration is therefore the effort required to achieve interoperability between disparate software programs.

Although often necessary, customized integration can be expensive and time consuming and can lead to fragile architectures that are burdensome to evolve. One of the goals of service orientation is to minimize the need for customized integration by shaping software programs (within a given domain) so that they are natively compatible. This is a quality referred to as intrinsic interoperability. The service orientation paradigm encompasses a set of specific design principles that are geared toward establishing intrinsic interoperability on several levels.

Intrinsic interoperability, as a characteristic of software programs that reside within a given domain, is key to realizing strategic benefits, such as increased cost-effectiveness and agility.

### **Shared services over specific-purpose implementations**

As just explained, service orientation establishes a design approach comprised of a set of design principles. When applied to a meaningful extent, these principles shape a software program into a unit of service-oriented logic that can be legitimately referred to as a service.

Services are equipped with concrete characteristics (such as those that enable intrinsic interoperability) that directly support the previously described target state. One of these characteristics is the encapsulation of multi-purpose logic that can be shared and reused in support of the automation of different business processes.

A shared service establishes itself as an IT asset that can provide repeated business value while decreasing the expense and effort to deliver new automation solutions. While there is value in traditional, single-purpose applications that solve tactical business requirements, the use of shared services provide greater value in realizing strategic goals of service-oriented computing (which again include an increase in cost-effectiveness and agility).

### **Flexibility over optimization**

This is perhaps the broadest of the value prioritization statements and is best viewed as a guiding philosophy for how to better prioritize various considerations when delivering and evolving individual services and inventories of services.

Optimization primarily refers to the fulfillment of tactical gains by tuning a given application design or expediting its delivery to meet immediate needs. There is nothing undesirable about this, except that it can lead to the aforementioned silo-based environments when not properly prioritized in relation to fostering flexibility.

For example, the characteristic of flexibility goes beyond the ability for services to effectively (and intrinsically) share data. To be truly responsive to ever-changing business requirements, services must also be flexible in how they can be combined and aggregated into composite solutions. Unlike traditional distributed applications that often were relatively static despite the fact that they were componentized, service compositions need be designed with a level of inherent flexibility that allows for constant augmentation. This means that when an existing business process changes or when a new business process is introduced, we need to be able to add, remove, and extend services within the composition architecture with minimal (integration) effort. This is why service composability is one of the key service orientation design principles.

### **Evolutionary refinement over pursuit of initial perfection**

There is a common point of confusion when it comes to the term “agility” in relation to service orientation. Some design approaches advocate the rapid delivery of software programs for immediate gains. This can be considered “tactical agility,” as the focus is on tactical, short-term benefit. Service orientation advocates the attainment of agility on an organizational or business level with the intention of empowering the organization, as a whole, to be responsive to change. This form of organizational agility can also be referred to as “strategic agility” because the emphasis is on longevity in that, with every software program we deliver, we want to work toward a target state that fosters agility with long-term strategic value.

For an IT enterprise to enable organizational agility, it must evolve in tandem with the business. We generally cannot predict how a business will need to evolve over time and therefore we cannot initially build the perfect services. At the same time, there is usually a wealth of knowledge that already exists within an organization’s existing business intelligence that can be harvested during the analysis and modeling stages of SOA projects.

This information, together with service orientation principles and proven methodologies, can help us identify and define a set of services that capture how the business exists and operates today while being sufficiently flexible to adapt to how the business changes over time.

**That is, while we value the items on the right, we value the items on the left more.**

By studying how these values are prioritized, we gain insight into what distinguishes service orientation from other paradigms. This type of insight can benefit IT practitioners

in several ways. For example, it can help establish fundamental criteria that we can use to determine how compatible service orientation is for a given organization or IT enterprise. It can further help determine the extent to which service orientation can or should be adopted.

An appreciation of the core values can also help us understand how challenging it may be to successfully carry out SOA projects within certain environments. For example, several of these prioritizations may clash head-on with established beliefs and preferences. In such a case, the benefits of service orientation need to be weighed against the effort and impact their adoption may have (not just on technology, but also on the organization and IT culture).

The upcoming guiding principles were provided to help address many of these types of challenges.

### **We follow these principles:**

So far, the manifesto has established an overall vision as well as a set of core values associated with the vision. The remainder of the declaration is comprised of a set of principles that are provided as guidance for adhering to the values and realizing the vision.

It's important to keep in mind that these are guiding principles specific to this manifesto. There is a separate set of established design principles that comprise the service orientation design paradigm and there are many more documented practices and patterns specific to service orientation and service-oriented architecture.

### **Respect the social and power structure of the organization.**

One of the most common SOA pitfalls is approaching adoption as a technology-centric initiative. Doing so almost always leads to failure because we are simply not prepared for the inevitable organizational impacts.

The adoption of service orientation is about transforming the way we automate business. However, regardless of what plans we may have for making this transformation effort happen, we must always begin with an understanding and an appreciation of the organization, its structure, its goals, and its culture.

The adoption of service orientation is very much a human experience. It requires support from those in authority and then asks that an IT culture adopt a strategic, community-centric mindset. We must fully acknowledge and plan for this level of organizational change in order to receive the necessary long-term commitments required to achieve the target state of service orientation.

These types of considerations not only help us determine how to best proceed with an SOA initiative, they further assist us in defining the most appropriate scope and approach for adoption.

## **Recognize that SOA ultimately demands change on many levels.**

There's a saying that goes: "Success is being prepared for opportunity." Perhaps the number one lesson learned from SOA projects carried out so far is that we must fully comprehend and then plan and prepare for the volume and range of change that is brought about as a result of adopting service orientation. Here are some examples.

Service orientation changes how we build automation solutions by positioning software programs as IT assets with long-term, repeatable business value. An upfront investment is required to create an environment comprised of such assets and an on-going commitment is required to maintain and leverage their value. So, right out of the gate, changes are required to how we fund, measure, and maintain systems within the IT enterprise.

Furthermore, because service orientation introduces services that are positioned as resources of the enterprise, there will be changes in how we own different parts of systems and regulate their design and usage, not to mention changes to the infrastructure required to guarantee continuous scalability and reliability.

## **The scope of SOA adoption can vary. Keep efforts manageable and within meaningful boundaries.**

A common myth has been that in order to realize the strategic goals of service-oriented computing, service orientation must be adopted on an enterprise-wide basis. This means, establishing and enforcing design and industry standards across the IT enterprise so as to create an enterprise-wide inventory of intrinsically interoperable services. While there is nothing wrong with this ideal, it is not a realistic goal for many organizations, especially those with larger IT enterprises.

The most appropriate scope for any given SOA adoption effort needs to be determined as a result of planning and analysis in conjunction with pragmatic considerations, such as the aforementioned impacts on organizational structures, areas of authority, and cultural changes that are brought about.

These types of factors help us determine a scope of adoption that is manageable. But for any adoption effort to result in an environment that progresses the IT enterprise toward the desired strategic target state, the scope must also be meaningful. In other words, it must be meaningfully cross-silo so that collections of services can be delivered in relation to each other within a pre-defined boundary. In other words, we want to create "continents of services," not the dreaded "islands of services."

This concept of building independently owned and governed service inventories within domains of the same IT enterprise reduces many of the risks that are commonly attributed to "big-bang" SOA projects and furthermore mitigates the impact of both organizational and technological changes (because the impact is limited to a segmented and managed scope). It is also an approach that allows for phased adoption where one domain service inventory can be established at a time.

**Products and standards alone will neither give you SOA nor apply the service orientation paradigm for you.**

This principle addresses two separate but very much related myths. The first is that you can buy your way into SOA with modern technology products, and the second is the assumption that the adoption of industry standards (such as XML, WSDL, SCA, etc.) will naturally result in service-oriented technology architecture.

The vendor and industry standards communities have been credited with building modern service technology innovation upon non-proprietary frameworks and platforms. Everything from service virtualization to cloud computing and grid computing has helped advance the potential for building sophisticated and complex service-oriented solutions. However, none of these technologies are exclusive to SOA. You can just as easily build silo-based systems in the cloud as you can on your own private servers.

There is no such thing as "SOA in a box" because in order to achieve service-oriented technology architecture, service orientation needs to be successfully applied; this, in turn, requires that everything we design and build be driven by the unique direction, vision, and requirements of the business.

**SOA can be realized through a variety of technologies and standards.**

Service orientation is a technology-neutral and vendor-neutral paradigm. Service-oriented architecture is a technology-neutral and vendor neutral architectural model. Service-oriented computing can be viewed as a specialized form of distributed computing. Service-oriented solutions can therefore be built using just about any technologies and industry standards suitable for distributed computing.

While some technologies (especially those based on industry standards) can increase the potential of applying some service orientation design principles, it is really the potential to fulfill business requirements that ultimately determines the most suitable choice of technologies and industry standards.

**Establish a uniform set of enterprise standards and policies based on industry, de facto, and community standards.**

Industry standards represent non-proprietary technology specifications that help establish, among other things, consistent baseline characteristics (such as transport, interface, message format, etc.) of technology architecture. However, the use of industry standards alone does not guarantee that services will be intrinsically interoperable.

For two software programs to be fully compatible, additional conventions (such as data models and policies) need to be adhered to. This is why IT enterprises must establish and enforce design standards. Failure to properly standardize and regulate the standardization of services within a given domain will begin to tear at the fabric of interoperability upon which the realization of many strategic benefits relies.

This principle not only advocates the use of enterprise design standards, it also reminds us that, whenever possible and feasible, custom design standards should be based upon and incorporate standards already in use by the industry and the community in general.

### **Pursue uniformity on the outside while allowing diversity on the inside.**

Federation can be defined as the unification of a set of disparate entities. While allowing each entity to be independently governed on the inside, all agree to adhere to a common, unified front.

A fundamental part of service-oriented architecture is the introduction of a federated endpoint layer that abstracts service implementation details while publishing a set of endpoints that represent individual services within a given domain in a unified manner. Accomplishing this generally involves achieving unity based on a combination of industry and design standards. The consistency of this unity across services is key to realizing intrinsic interoperability.

A federated endpoint layer further helps increase opportunities to explore vendor-diversity options. For example, one service may need to be built upon a completely different platform than another. As long as these services maintain compatible endpoints, the governance of their respective implementations can remain independent. This not only highlights that services can be built using different implementation mediums (such as EJB, .NET, SOAP, REST, etc.), it also emphasizes that different intermediary platforms and technologies can be utilized together, as required.

Note that this type of diversity comes with a price. This principle does not advocate diversification itself – it simply recommends that we allow diversification when justified, so that “best-of-breed” technologies and platforms can be leveraged to maximize business requirements fulfillment.

### **Identify services through collaboration with business and technology stakeholders.**

In order for technology solutions to be business-driven, the technology must be in synch with the business. Therefore, another goal of service-oriented computing is to align technology and business. The stage at which this alignment is initially accomplished is during the analysis and modeling processes that usually precede actual service development and delivery.

The critical ingredient to carrying out service-oriented analysis is to have both business and technology experts working hand-in-hand to identify and define candidate services. For example, business experts can help accurately define functional contexts pertaining to business-centric services, while technology experts can provide pragmatic input to ensure that the granularity and definition of conceptual services remains realistic in relation to their eventual implementation environments.

### **Maximize service usage by considering the current and future scope of utilization.**

The extent of a given SOA project may be enterprise-wide or it may be limited to a domain of the enterprise. Whatever the scope, a pre-defined boundary is established to encompass an inventory of services that need to be conceptually modeled before they can be developed. By modeling multiple services in relation to each other we essentially establish a blueprint of the services we will eventually be building. This modeling exercise is critical when attempting to identify and define services that can be shared by different solutions.

There are various methodologies and approaches that can be used to carry out service-oriented analysis stages. However, a common thread among all of them is that the functional boundaries of services be normalized to avoid redundancy. Even then, normalized services do not necessarily make for highly reusable services. Other factors come into play, such as service granularity, autonomy, state management, scalability, composability, and the extent to which service logic is sufficiently generic so that it can be effectively reused.

These types of considerations guided by business and technology expertise provide the opportunity to define services that capture current utilization requirements while having the flexibility to adapt to future change.

### **Verify that services satisfy business requirements and goals.**

As with anything, services can be misused. When growing and managing a portfolio of services, their usage and effectiveness at fulfilling business requirements need to be verified and measured. Modern tools provide various means of monitoring service usage, but there are intangibles that also need to be taken into consideration to ensure that services are not just used because they are available, but to verify that they are truly fulfilling business needs and meeting expectations.

This is especially true with shared services that shoulder multiple dependencies. Not only do shared services require adequate infrastructure to guarantee scalability and reliability for all of the solutions that reuse them, they also need to be designed and extended with great care to ensure their functional contexts are never skewed.

### **Evolve services and their organization in response to real use.**

This guiding principle ties directly back to the “Evolutionary refinement over pursuit of initial perfection” value statement, as well as the overall goal of maintaining an alignment of business and technology.

We can never expect to rely on guesswork when it comes to determining service granularity, the range of functions that services need to perform, or how services will need to be organized into compositions. Based on whatever extent of analysis we are able to initially perform, a given service will be assigned a defined functional context and will contain one or more functional capabilities that likely involve it in one or more service compositions.

As real world business requirements and circumstances change, the service may need to be augmented, extended, refactored, or perhaps even replaced. Service orientation design principles build native flexibility into service architectures so that, as software programs, services are resilient and adaptive to change and to being changed in response to real world usage.

### **Separate the different aspects of a system that change at different rates.**

What makes monolithic and silo-based systems inflexible is that change can have a significant impact on their existing usage. This is why it is often easier to create new silo-based applications rather than augment or extend existing ones.

The rationale behind the separation of concerns (a commonly known software engineering theory) is that a larger problem can be more effectively solved when decomposed into a set of smaller problems or concerns. When applying service orientation to the separation of concerns, we build corresponding units of solution logic that solve individual concerns, thereby allowing us to aggregate the units to solve the larger problem in addition to giving us the opportunity to aggregate them into different configurations in order to solve other problems.

Besides fostering service reusability, this approach introduces numerous layers of abstraction that help shield service-comprised systems from the impacts of change. This form of abstraction can exist at different levels. For example, if legacy resources encapsulated by one service need to be replaced, the impact of that change can be mitigated as long as the service is able to retain its original endpoint and functional behavior.

Another example is the separation of agnostic from non-agnostic logic. The former type of logic has high reuse potential if it is multi-purpose and less likely to change. Non-agnostic logic, on the other hand, typically represents the single-purpose parts of parent business process logic, which are often more volatile. Separating these respective logic types into different service layers further introduces abstraction that enables service reusability while shielding services, and any solutions that utilize them, from the impacts of change.

### **Reduce implicit dependencies and publish all external dependencies to increase robustness and reduce the impact of change.**

One of the most well-known service orientation design principles is that of service loose coupling. How a service architecture is internally structured and how services relate to programs that consume them (which can include other services) all comes down to dependencies that are formed on individually moving parts that are part of the service architecture.

Layers of abstraction help ease evolutionary change by localizing the impacts of the change to controlled regions. For example, within service architectures, service facades can be used to abstract parts of the implementation in order to minimize the reach of implementation dependencies.

On the other hand, published technical service contracts need to disclose the dependencies that service consumers must form in order to interact with services. By reducing internal dependencies that can affect these technical contracts when change does occur, we avoid proliferating the impact of those changes upon dependent service consumers.

**At every level of abstraction, organize each service around a cohesive and manageable unit of functionality.**

Each service requires a well-defined functional context that determines what logic does and does not belong within the service's functional boundary. Determining the scope and granularity of these functional service boundaries is one of the most critical responsibilities during the service delivery lifecycle.

Services with coarse functional granularity may be too inflexible to be effective, especially if they are expected to be reusable. On the other hand, overly fine grained services may tax an infrastructure in that service compositions will need to consist of increased quantities of composition members.

Determining the right balance of functional scope and granularity requires a combination of business and technology expertise, and further requires an understanding of how services within a given boundary relate to each other.

Many of the guiding principles described in this manifesto will help in making this determination in support of positioning each service as an IT asset capable of furthering an IT enterprise toward that target state whereby the strategic benefits of service-oriented computing are realized.

Ultimately, though, it will always be the attainment of real world business value that dictates, from conception to delivery to repeated usage, the evolutionary path of any unit of service-oriented functionality.

*Acknowledgements: I wrote the content on this page on the weekend of November 21-22, 2009 for the Next Generation SOA book that was still under development at the time. I'd like to thank Prentice Hall for allowing this content to be openly published on this site as a supplement to the original SOA Manifesto. I'd also like to thank the working group members and those from the SOAPatterns.org community that provided feedback and assistance with these annotations. Many of the working group members have published their own articles, blogs, and papers about the SOA Manifesto and I encourage you all to seek these out for further commentary and insights.*

*- Thomas Erl*