zapthink
white paper

SOA Quality across the Service Lifecycle

ZapThink, LLC • 108 Woodlawn Road • Baltimore, MD 21210 • info@zapthink.com • www.zapthink.com

SOA Quality across the Service Lifecycle

August 2007

Analyst: Jason Bloomberg

Abstract

As organizations move beyond straighforward implementations of Web Services to the more complex world of Service-Oriented Architecture (SOA), maintaining the quality of the implementation in the face of changing business requirements becomes an increasingly difficult challenge. In the final analysis, quality represents how well a system meets the needs of the business, so when the business case for SOA calls for business agility, SOA quality means meeting business requirements as those requirements are in an ongoing, continual state of change.

As a result, SOA quality extends well beyond traditional quality assurance tasks to cover the full Service lifecycle, and encompasses both SOA management and governance into a broad set of capabilities that any organization must implement in order to be successful with their SOA initiatives.

I. The Promise and Problems of SOA

Many people once considered Information technology (IT) to be an unpleasant but necessary expense for the enterprise. But now, for many industries, IT has become an essential part of their business. Companies rely upon IT to provide better value for customers, greater power to employees, and improved agility to the organization as a whole. Companies whose IT environments lack flexibility or cost effectiveness risk losing their competitive edge in the marketplace to firms that can creatively leverage their technology assets for competitive advantage.

This increasing reliance on IT comes in the face of burgeoning complexity and business change. Companies that adapt to an ever-changing business environment are able to compete more effectively and thrive in any business climate. Such companies are particularly adept in tough economic times, often finding opportunities in the midst of chaos.

IT is frequently a significant barrier to companies' ability to respond to change.

Instead of being an enabler, however, IT is frequently a significant barrier to companies' ability to respond to change, for two reasons. First, technology has become so intertwined with the organization's existing business processes that changing a process requires changing the technology—a complicated, costly, and error-prone endeavor. Second, because technology change is often expensive, difficult, and complex, the inherent quality risk of changing technology becomes a limiting factor for the business.

It is to address this disconnect between the agility the business requires and the agility that IT can provide that gives rise to the movement toward Service Orientation. Service Orientation is a business approach that leverages IT resources as flexible, business-oriented Services. Services abstract the underlying complexity of the IT environment, providing greater power and flexibility to the business. Service Orientation has the power to increase competitiveness in the face of today's ever-changing business environment, and once businesses realize the transformative power of this critically important business concept, they will be in the position to deal with ongoing, often unpredictable, change.

Many organizations leverage SOA to reduce the cost of integration, increase asset reuse, and improve customer visibility and overall business transparency, as they build the business case for agility.

The core enabler of Service Orientation is *Service-Oriented Architecture* (SOA), a set of best practices for organizing and managing IT resources and people to build and support such Services. Today, the majority of enterprises and government organizations are somewhere on their SOA roadmap. While business agility is the most strategic business benefit of SOA, however, it is also the most difficult to justify. Instead, many organizations leverage SOA to reduce the cost of integration, increase asset reuse, and improve customer visibility and overall business transparency, as they build the business case for agility.

SOA does not eliminate complexity—it abstracts *the complexity, providing a flexible, simplified set of Services.*

Complexity behind the scenes: the technical challenge of heterogeneity

One of the reasons that achieving business agility with SOA is such a challenge for many organizations is the technical challenge of heterogeneity. Today's enterprise IT environments are enormously complex, and it is that complexity more than any other cause that leads to the inflexibility that the business wishes to address. And yet, SOA does not actually eliminate complexity—it *abstracts* the complexity, providing a flexible, simplified set of Services to the business that overlays the unavoidable technical complexity.

At the core, IT has always dealt with underlying complexity, through the power of abstraction. In the world of IT, abstraction is a way to simplify the complexities of the technology with simple, yet powerful representations. Beneath the abstraction layer, there remains the complexity that IT deals with today. But due to the power of loose coupling, the Services available to the business provide whatever value the business requires from them. It's vitally important to SOA that we place such Services into the business context.

As with any abstraction, however, there is no magic here. To build such powerful services requires sophisticated governance, management, and an overall focus on quality. After all, while it's simple to talk about loose coupling—where it's possible to independently control and change Service providers and consumers without impacting the other—that loose coupling depends upon high quality, well-managed and governed SOA infrastructure. In fact, companies will not be able to break down the IT/business disconnect until they solve this underlying governance/quality/management convergence problem.

Full-lifecycle quality growing in importance

Quality means far more than simply reducing defects. Fundamentally, quality means building something that meets the requirements of its users, now and into the future. Being defect-free is a necessary, but by no means sufficient criterion for a quality product. Software quality is no different. While many software quality assurance efforts focus on eliminating bugs, the bug-hunting process is only the starting point for software quality.

The real challenge with software quality, as with any other quality effort, is in guaranteeing that the software meets the requirements set out for it, because only by providing this guarantee can IT establish trust. In an ideal world, quality assurance (QA) personnel would simply take the requirements document, use it to build a test plan, and run tests against that plan. Once the project passes all the tests, it's ready to go live. But in the real world, requirements continue to evolve, both during projects as well as once the projects are complete. And there's nothing worse than evolving requirements for throwing a wrench in the most carefully laid QA plans.

Environments of continually changing business requirements, of course, are the perfect breeding ground for SOA. SOA leverages a metadata-driven Service abstraction to provide greater power and flexibility to business users, with the clear purpose of enabling IT to respond to changing requirements in an agile manner. This core agility benefit of SOA collapses like a house of cards, however, if the Services or the applications that consume and compose them are of poor quality or behave in an unpredictable manner.

Not all organizations require such full-lifecycle quality practices, especially when they have no particular requirement for agility. The defining moment for stronger quality practices arises when there is either a high rate of change in the underlying systems, or a high degree of complexity and change in the business rules or requirements.

The SOA story touches upon many challenges like these—including loose coupling, trust, governance, and management, to name a few. But in the final analysis, SOA quality becomes the primary organizing principle for actually getting this stuff to work—what does it actually take to enable IT to meet the changing needs of the business. SOA quality becomes the thread that enables the business to trust technology to provide the agility it requires to meet requirements now and into the future.

II. SOA Quality: Beyond Web Services

Quality should be first and foremost in the mind of everyone on a software project.

Traditionally, software testing has been the ugly duckling of software development. Long thought a necessary evil to be relegated for the end of a project, testing often gets the budgetary axe in software projects that are experiencing cost overruns. But just like the swan in the story, testing is actually a key element of software quality—and quality, broadly defined, is the ability of a system to do what it's supposed to do. Therefore, quality should be first and foremost in the mind of everyone on a software project.

The software industry as a whole has come to realize the importance of quality, and in most software organizations today, testing is an important, integral part of the software development process. Furthermore, the development processes themselves are in the process of transforming, as the traditional "put testing last" waterfall methodology gives way to less risky, more proactive iterative software development processes.

Web Services testing in the context of SOA

Today, however, the software industry is transitioning to the loosely coupled world of SOA, where Services are at a higher level of abstraction than Web Services. Yet for many organizations, the primary use for Web Services is to simplify and reduce the cost of integration within the enterprise. Such straightforward applications of Web Services—wrapping components with SOAP interfaces so that they can exchange XML-based messages with other components—are relatively straightforward to test with today's software testing tools and techniques.

As organizations move beyond simple Web Services implementations to considerations of implementing SOA, however, many testing issues will arise beyond simple Web Services testing, which include the following capabilities:

- ***Testing the publish, find, and bind capabilities within SOA*** – three fundamental characteristics of SOA are the publish, find, and bind capabilities of the constituent Web Services: a Web Service provider first publishes the WSDL file for the Service in a UDDI registry, where Web Service consumers can search for it and locate its WSDL file. The consumers then bind to the Service based upon the WSDL file in the registry.
- ***Web Services orchestration testing*** – One of the most powerful uses of Web Services is orchestration: combining many fine-grained Web Services into larger, coarse-grained Services. Such orchestration can be either recursive (Services made up of several Services that in turn may also be made up of Services) or sequential (a series of Services arranged to follow a business process), or some combination thereof. Such orchestrations of Services typically involve more than one company, for example, when they represent a B2B transaction. Testing orchestrated Web Services is a form of system testing, only with Web

Services, there will be combinations of synchronous and asynchronous services, and many of the Services may be dynamically described and discovered.

- *Service-level agreement (SLA) and Quality of Service (QoS) monitoring* – Because IT managers can install and upgrade Web Services on an *ad hoc* basis, there will be a greater need to test Web Services during runtime than in a traditional IT environment. Traditionally, IT management is the province of operations, while software testing belongs to development. This distinction will gradually blur as Service-oriented environments become more prevalent. Therefore, Web Services testing tools should have runtime testing capabilities in addition to their design time capabilities, verifying that Web Services, individually and in concert, are performing the way they should.
- *Full lifecycle SOA testing* – a further ramification of the *ad hoc* upgrade capability of the Service-oriented environment is that rolling out new versions of Services—especially when they are combined into complex orchestrations—will be complex and risky. For such *ad hoc* changes to be successful, therefore, the enterprise must have testing tools that can test the new versions of individual Web Services in the production environment.

As companies embark on their SOA initiatives, quality in the face of change should be a top priority.

As companies embark on their SOA initiatives, therefore, quality in the face of change should be a top priority. More often than not, however, quality receives short shrift in many such projects, especially when an organization is taking a bottom-up approach to SOA that begins by building Web Services interfaces to their existing legacy systems. For such organizations, their SOA project is light on architecture and heavy on software development, as they hammer out the details of their Service interfaces. As a result, they typically limit their QA efforts to the testing of those Service interfaces.

As organizations go beyond the straightforward use of Web Services to implementing SOA, the SOA quality tasks extend well beyond testing Web Services as simple software interfaces. SOA introduces the concept of a Service at a higher level of abstraction that requires the consideration of numbers of interdependent Services that evolve over time. SOA quality thus becomes a full lifecycle activity that goes well beyond Web Services testing.

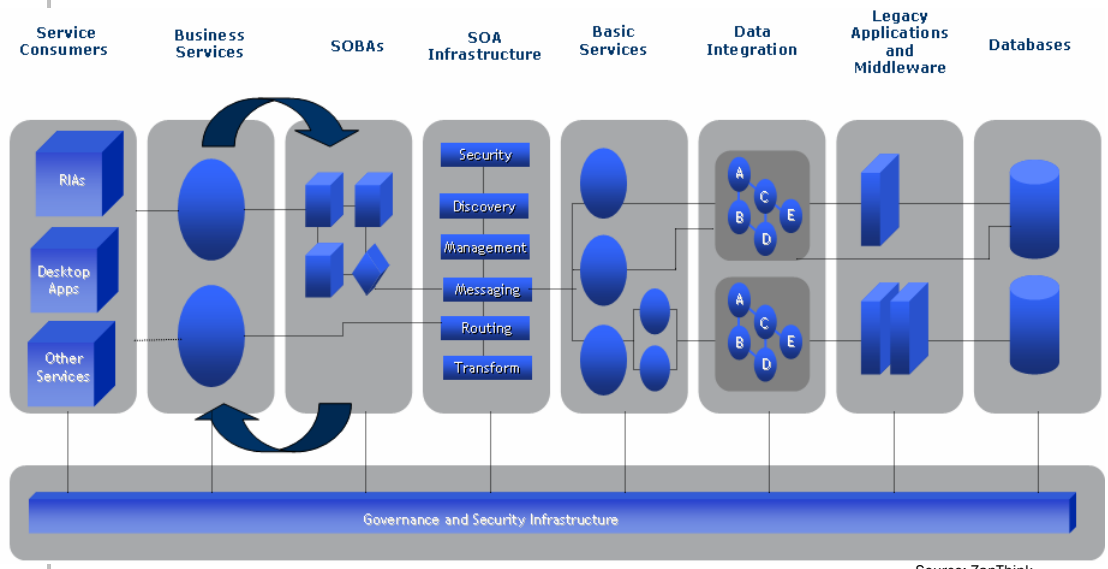
The increasing sophistication of SOA testing tools

Web Services testing is the most basic capability of SOA testing tools. In fact, many tools on the market that claim to be SOA testing tools are really little more than Web Services testing tools that enable companies to put their Web Services (as well as Service consumers) through their paces in an effort to reduce defects or ensure that the Services comply with the structural requirements set out for them. Many such tools are updated versions of Web page testing tools that generalize the Web interface, basically considering a Web Service to be a Web page without the user interface. While such Web Services testing tools serve an important role, they are insufficient for providing the QA required for true SOA implementations.

More sophisticated SOA testing tools take into account the fact that Services are more than standards-based interfaces—rather, they're abstractions of capabilities from multiple, disparate sources. Such tools approach the SOA quality problem as an integration testing challenge, and it's not surprising that many of such tools currently on the market have evolved from integration testing products. These tools simulate Service requests and other events as they wind

their way through the Service interface and underlying middleware, applications, and data sources, uncovering subtle defects that arise from the complex interactions among the various moving parts in today's distributed systems. While such integration testing is a critical part of any SOA quality regimen, it is fundamentally a design time activity, as is Web Service testing. Neither approach provides much value after the Services go live. The integration testing challenge is apparent in the figure below, which illustrates a typical SOA environment.

Typical SOA Environment



Source: ZapThink

The most sophisticated of SOA quality tools take into account the full Service lifecycle—design time, runtime, and change time.

The most sophisticated of SOA quality tools take into account the full Service lifecycle—design time, runtime, and change time. No longer is it sufficient to run a project through acceptance testing immediately before launching it into production, because SOA implementations are by their very nature continually changing. Instead, SOA quality must be an ongoing process that continually confirms that the existing configuration of Services meets the business requirements *du jour*.

The tooling necessary to implement such advanced quality measures must focus on testing SOA metadata, because metadata are at the core of any SOA implementation's ability to respond dynamically to changing business requirements. The changes that occur during change time are metadata changes, including Service-Oriented Business Application (SOBA) configuration changes, policy changes, and Service contract changes. Today's most advanced SOA quality tools must provide for the testing of changes to these metadata in a production environment.

The agile requirement of SOA quality

Clearly, the ponderous "design, then develop, then test, then launch" process from the sequential waterfall software development methodology will be entirely inappropriate for the fully realized Service-oriented environment. Instead, software development groups will by necessity have to take what is now known as an agile approach to testing and development: work with the customer, tackle just what needs to be done, write a test, code as a team, pass the test, repeat

until finished. Testing before coding will move from being "extreme" (as in the popular agile development methodology Extreme Programming) to being the only cost-effective method for dealing with the dynamic nature of distributed computing.

Taking the agile approach to testing is the only approach to software testing that makes sense for SOA implementations.

Taking the agile approach to testing, in fact, appears to be the only approach to software testing that makes sense for agile SOA implementations. A system test would never be comprehensive and complete if you could only conduct it as a scheduled part of the design process, because you haven't determined the individual Services that make up that system during the design. In the agile approach, in contrast, testing begins by automating the collection and representation of new user requirements. The tests themselves, as well as the resulting code, develop iteratively, so that when the code is complete, the test is as well—and the code adapts to pass the test at each iteration. If each individual Service has its own automated test, then orchestrating Services also includes orchestrating the tests.

Keep in mind that typical business requirements for software are quite coarse grained, by definition. As a development team works with users to define coarse-grained requirements and write the corresponding coarse-grained tests, the testing tools that the development team uses must be able to automate the coordination of the appropriate tests for all existing and new Web Services, on all levels of granularity.

In fact, the whole concept of a software development lifecycle, where development teams design, build, launch, and eventually retire large software projects, will fall by the wayside as companies realize the advantages of orchestrating loosely coupled Services into coarse-grained business Services. The iterative methodologies that were such an improvement over the earlier waterfall methodology must now change as well, and become agile. Agile approaches to software development will by necessity become the only cost-effective, practical way of conducting the practice of creating software in the fully realized Service-oriented environment.

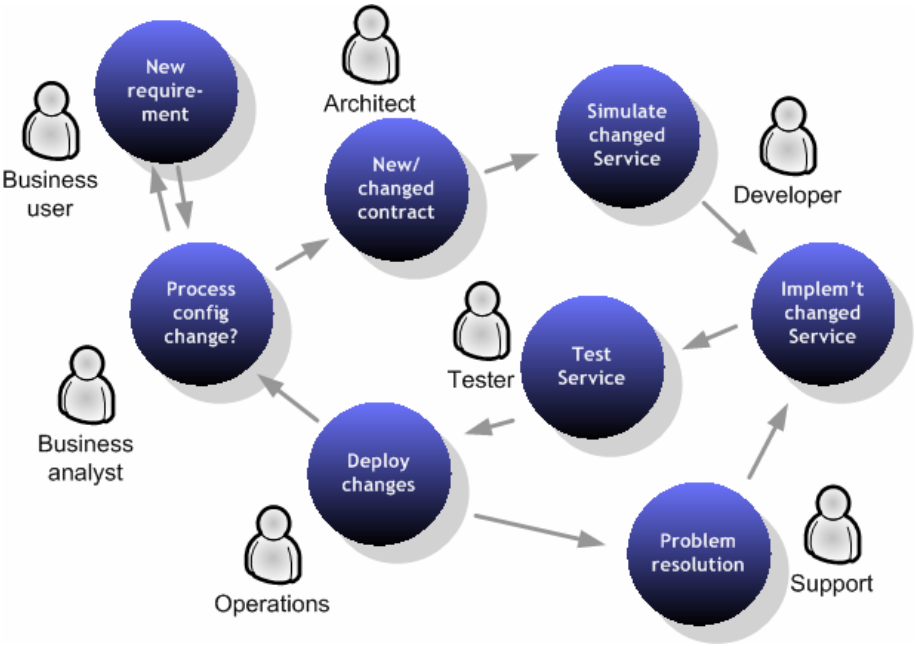
Full lifecycle SOA quality

The more mature the SOA implementation is, the more impractical traditional quality assurance approaches are likely to be.

Perhaps the greatest SOA quality challenge, however, involves maintaining quality throughout the Service lifecycle, especially once the SOA implementation is in place, as shown in the figure below. The problem is, the more mature the SOA implementation is—that is, the better the Service abstraction maintains an agile separation between business users and the underlying IT capabilities—the more impractical traditional QA approaches are likely to be. In many of today's IT shops, there are separate, identical QA and production environments. QA personnel can load any new or changed code into the QA environment, and test it to their heart's content before giving it the thumbs up for promotion of changes to the production environment.

In a mature SOA environment, it's practically impossible to maintain a useful duplicate of the running system, because Services, configurations, and associated metadata continually change. As a result, maintaining a parallel QA environment rapidly becomes an exercise in futility. Furthermore, it's clear from the figure below that SOA quality is a collaborative effort that involves many different participants across the organization.

Quality Throughout the Service Lifecycle



Source: ZapThink

The solution to this quality conundrum is to test new and changed Services and Service configurations in the live, production environment. The only way to ensure that all aspects of the new configuration continues to meet the requirements set out for it is to run test messages through production Services. Now, saying you should test in production is tantamount to proposing rewiring your house with the power on—it's possible, but you have to be especially careful, know what you're doing, and plan ahead. In the case of SOA, planning ahead means that Services (as well as Service consumers) must be able to support a testing mode.

There are a few important notes about running the QA process through a production system. First, the QA process is policy-driven. As a result, the testing process is itself Service-oriented, which goes a long way to satisfying the meta-requirement of SOA. Second, it's resilient. Even if a fully tested Service still fails in production, the QA process responds in a way that minimizes the business impact, and thus maximizes the loose coupling of the Services. But most importantly, SOA quality requires planning ahead. Architects must plan for test modes and deprecation policy support as an essential part of designing Services.

III. The Meta-Requirement of Agility

The meta-requirement for SOA is the requirement that the implementation of the architecture must be able to satisfy future requirements even as they continually evolve.

Simply increasing the sophistication of your testing tools doesn't mean your SOA will be of any higher quality. In fact, if a QA team treats a SOA project as though it were a traditional software project, its quality will likely suffer, because traditional projects don't have the business requirement of agility—what ZapThink calls the "meta-requirement" for SOA, namely the requirement that the implementation of the architecture must be able to satisfy future requirements even as they continually evolve. A core challenge of SOA, after all, is building for change. If you had good reason to believe today's requirements were permanent, you probably wouldn't bother with the expense and complexity of SOA. Testing for

today's requirements without testing for this meta-requirement leaves an enormous hole in the QA process.

However, it doesn't make sense to expect that the IT organization would be capable of building systems that could deal with entirely arbitrary change, since such a requirement would be prohibitively expensive to satisfy. Instead, each organization will have to decide for itself precisely how much they can invest to properly scope the level of flexibility they require their SOA implementation to have. While deciding on your agility meta-requirement is an essential part of your SOA planning, even more important for the long-term success of your SOA initiative is in ensuring that your SOA implementation conforms to that meta-requirement over time. In other words, guaranteeing that your SOA meets the needs of the business over time is a core measure of quality, and as such, SOA quality assurance must go beyond traditional software quality management and address the meta-requirement of agility.

Change time quality

Traditional software quality management essentially consists of design time and deployment time activities. Basically, given the requirements, make sure that the software is as defect-free as possible given budget and schedule constraints, and then make sure that it meets the requirements set out for it when you deploy it. That basic approach to quality is fine for organizations that know in advance what their requirements are, when those requirements are stable, and when the goal is simply to build software that meets those requirements.

Such assumptions fail to address IT environments where requirements aren't fully developed and they change over time. Ideally, software should be responsive to changes in requirements without extensive additional rework. SOA is a particularly effective approach in such situations, and the broad recognition that the build-to-today's-requirements approach to software is no longer effective is one of the primary motivations for SOA.

SOA enables organizations to effect requirements changes via declarative reconfiguration of Service metadata.

While many existing SOA management tools on the market handle runtime SOA management, it is not the intention of these tools to handle quality management in the face of ongoing requirements changes—at *change time*. Properly implemented, SOA enables organizations to effect requirements changes via declarative reconfiguration of Service metadata. Therefore, change time quality management focuses on metadata, and how well those metadata satisfy the requirements that apply during change time. Such requirements fall into two categories: ongoing, day-to-day requirements that reconfiguration can address, and the broader meta-requirement of agility.

The intractability of change-time metadata quality assurance results from the open-ended nature of reconfiguration in SOA. If architects fail to fully plan for this problem ahead of time, the sorts of changes users might introduce over time could be entirely unpredictable and unmanageable. Fundamentally, SOA encourages user empowerment, which in the absence of adequate governance can lead to anarchy. The important point to keep in mind is that change-time metadata quality assurance should be handled as a matter of policy. Apply Service-oriented principles to change-time quality assurance in order to create, communicate, and enforce policies for metadata-based changes. Then you can treat those policies like other policies that your SOA governance infrastructure deals with.

Exploring the meta-requirement of agility

As the enterprise architecture team sits down to plan your SOA initiative, they should be asking questions about the specific levels of agility the organization

requires from the SOA implementation. In particular, they should discover the answers to the following questions:

- What are the business requirements for Service reuse? Is the business looking to achieve a certain cost savings or other metric for reuse?
- What are the requirements for loose coupling? Break this question down into multiple levels: semantic, data, message protocol, wire protocol, etc.
- What is the user context for the implementation? For example, how many users will the Services need to support? Are they all internal users, or are you allowing users from outside your organization? What users will be allowed to make configuration changes, and under what circumstances?
- What are the metapolicies? In other words, what policies are in place that govern how the organization should be able to create and enforce policies?

It's important to note first of all that the end product of each of these questions should be a set of policies. Secondly, it's also significant that questions like these don't focus on requirements for specific functionality or behavior of the software you're building as in a traditional project. On the contrary, these questions all dwell on issues of agility—just how agile your organization wants to be, but possibly even more importantly, in what areas is it OK to be less than fully agile. Properly scoping the constraints on the SOA initiative that your organization accepts can save substantial money and time, and can also lead to framing the discussion of change-time quality policies.

Governance and the meta-requirement of agility

SOA governance streamlines the part of the governance framework that lends itself to automation, and provides better tools for people to handle the rest.

Even though considering change-time quality to be a governance issue reduces the risks inherent in the user empowerment benefit of SOA, it's important to keep in mind that governance is not fully automatable. In fact, most governance activities are human: training, organizational management, and other human communications activities. SOA governance streamlines the part of the governance framework that lends itself to automation, and provides better tools for people to handle the rest. But no matter how sophisticated today's SOA governance solutions become, the human element will always remain.

As an example of this principle, say an organization has a policy that a manager must review all Service compositions before going into effect. That policy is clearly a change-time quality policy, especially if there are associated policies that guide what managers should be looking for when they conduct such reviews. A tool can ease the implementation of this policy, but cannot take the manager out of the equation.

A second, more significant example concerns the potential infinite regression: if metadata quality is a matter of policy, including the quality of the policy metadata themselves, then how can you ensure the quality of those policies that apply to your quality policies? The answer is surprisingly simple: quality at the metapolicy level is up to people to provide. Say you have a policy that all Service contracts must conform to a particular Service specification that your organization has hammered out. Automating the enforcement of that policy is straightforward, but ensuring the quality of that policy is traditionally a human activity: read through the meta-requirements list and see if that policy is there, and make sure that people understand it properly. Such activities are not only time-consuming for people, but they are difficult to automate.

IV. Governance, Quality, and Management: The Full Spectrum of SOA Quality

Considering change time quality assurance to be a matter of policy enforcement completes the SOA quality tooling picture: testing tools for design time quality, management tools for runtime quality, and governance tools for change time quality. Yet while the marketplace recognizes each of these critical areas of SOA governance, quality, and management, only recently have we realized that each of these distinct market segments in their own right are really different aspects of same problem: making the challenge of loose coupling a reality. In combination, SOA governance, quality, and management are all part of the same spectrum of capabilities that can make the perceived difficulty of loose coupling in a continuously changing IT and business environment a reality, as shown in the figure below.

The SOA Governance/Quality/Management Triangle

Source: ZapThink LLC

The connection between runtime quality and management

One part of this combined set of capabilities is the connection between runtime SOA quality and SOA management. It makes very little difference if one Service works in isolation if some aspect of how it participates in Service compositions, or how some change to metadata impacts the performance of the system as a whole. In essence, unit testing an individual Service implementation is wholly inadequate to determining whether the Service actually performs in a composite environment of metadata-controlled Services.

The only way to effectively ensure SOA quality is to do so continuously.

The only way to effectively ensure SOA quality is to do so continuously, measuring the quality not only of a discrete, atomic Service, but also all related metadata, composition logic, policy, and underlying schema continuously in production. However, within the context of SOA the architecture *is* the business, and since the business is continuously changing, a QA model that requires duplication of the environment in order to ensure quality will be enormously expensive, impossible to manage, and ineffective.

Making runtime quality actually work in production requires a mechanism to isolate failures from having recursive and unpredictable effects by implementing test modes as matters of Service contract and policy, and minimizing the side-

effects of Services in a test mode such to avoid any unnecessary commitment of data or action. The ideas of policy-driven test and runtime enforcement of quality overlap those of runtime Service management. Many SOA management solutions provide policy enforcement, exception management, failure recovery, and root-cause analysis. Pairing the capabilities of runtime quality tools that facilitate the process of versioning with solutions that minimize the impact of those changes is a natural fit.

Furthermore, there is a management-quality feedback loop that exists between tools and approaches to management that provide visibility when systems are approaching an undesired state and mechanisms that allow incremental testing and quality management of the system as a whole. This feedback helps to guarantee loose coupling by making sure that any Service-related changes don't break things, which is a fundamental requirement of loose coupling.

The connection between SOA governance and management

Likewise, there is a connection between SOA governance and management that facilitates loose coupling. SOA governance has three distinct, but related parts: design time governance that provides rules and policies for creating, exposing, and consuming Services, runtime governance that governs the behavior of Services in production and the performance of the architecture as a whole, and change time governance that details how organizations can effect changes in the overall system with the least disruption to the existing business and its policies.

SOA management offerings can address aspects of policy enforcement, rules-based routing and decision making, and exception handling. As such, SOA management tools can enforce policies at runtime that governance tooling manages at design time. While SOA governance tooling such as registries and repositories serve as systems of record to manage Service metadata, SOA management approaches ensure that Services in production comply with the policies in effect. SOA management tooling can also detect and prevent the occurrence of rogue Services and inspect Service interaction, further enabling the value of SOA governance approaches. In addition, runtime SOA management helps with change-time governance issues by enforcing governed and approved changes in the distributed environment while minimizing quality and performance issues.

Furthermore, effective SOA governance requires effective management to provide the visibility runtime systems require to feed back into the governance process. This management-governance feedback loop not only helps companies govern their overall SOA efforts, but also provides effective control and management without overly constricting the agility of the architecture. This feedback loop helps to guarantee loose coupling by making sure that business requirements for change don't have an adverse impact on the behavior of the whole system, by decoupling the logic of the business from the implementation as it exists at that point.

The connection between SOA governance and quality

Implicit in the discussion above is the relationship between change time management and governance. Organizations must manage and govern change effectively so that no change has a chaotic effect on the complex environment of Services. This challenge means not only catching and preventing failure through management approaches at runtime, but also staging and testing those changes through runtime quality and testing approaches. Design time governance also requires that companies enforce Service development practices before uncontrolled Services permeate the IT environment. Effective integration

between governance and quality approaches makes that enforcement possible by providing the visibility into deviations from established policy and methods for limiting the spread of Services that are non-compliant.

The challenge of maintaining continuous quality in a continually changing system is an aspect of maintaining effective governance, and as such, the combination of SOA governance and quality tooling and approaches serve to make that problem more manageable.

The challenge of maintaining continuous quality in a continually changing system is an aspect of maintaining effective governance, and as such, the combination of SOA governance and quality tooling and approaches serve to make that problem more manageable. SOA quality tools provide feedback to governance systems by indicating how changing policies impact the overall system, and likewise, governance systems and approaches feed into the quality lifecycle by providing continually changing constraints that impact Services at design and runtime. This quality-governance feedback loop furthers the realization of loose coupling by making sure that any design time change has no impact on running systems, and that Service consumers do not need to hardcode governance rules.

The emergence of the SOA GQM suite

It is imperative that companies place significant effort on the governance/quality/management aspect of SOA infrastructure, as these capabilities are far more critical to achieving the business benefits of SOA than integration middleware, for example. We're not saying that SOA integration infrastructure, such as an Enterprise Service Bus, is unnecessary, but rather that it is not sufficient to guarantee the loose coupling and composability in an environment of continual change that companies desire of their SOA implementation.

Therefore, companies should focus on the kind of SOA-specific enablement technology that is central to the goals of SOA: the SOA Governance/Quality/Management (GQM) suite. The suite might not necessarily be a single-vendor product solution, even though it seems the market might actually be converging in this manner. Regardless of whether it is a single-vendor solution or a collection of best-of-breed applications, the SOA GQM suite is the collection of tools and capabilities that facilitate all three feedback loops that this paper discusses.

This complete feedback loop that links the three loops above is already becoming a reality in successful SOA projects. Right now, companies should focus on addressing each part of the GQM suite by making sure that they are implementing all of the feedback loops. Whether through home-grown solutions, best-of-breed vendor solutions, or packaged offerings that aim to provide the whole suite of offerings, companies must properly address their GQM needs if they ever hope to realize the benefits of SOA.

Companies must properly address their governance/quality/management needs if they ever hope to realize the benefits of SOA.

V. The ZapThink Take

For anyone who still confuses quality with testing, it's important to remember that testing, after all, is quality assurance, and the importance of quality remains absolute. Architectures, computing approaches, platforms, languages—all of these may change, but quality retains its primary importance. Vendors and enterprises that lose track of the fundamental tenet of quality, that software is there to meet the needs of the people that use it, do so at their own peril. If SOA does not meet user requirements—in other words, it lacks sufficient quality—then there can be no trust in the movement toward loosely coupled, open standards-based distributed computing. In such instances, SOA initiatives may falter, or even fail altogether.

SOA quality, however, goes well beyond Web Services testing into the full lifecycle consideration of interdependent, continually changing abstracted Services. As a result, SOA quality cannot stand alone—to cover design time,

runtime, and change time, organizations require fully coordinated governance/quality/management capabilities. Investing in such GQM tooling in prior architectural and technological evolutions of IT simply was not necessary in an environment of tightly-coupled, proprietary systems, and lack of composable systems. In such environments, testing is a QA activity that takes place between development and deployment, management relegated to the post-deployment phase, and then only in a monitoring capacity, and governance an afterthought altogether. However, the compelling value of loose coupling and composition that SOA represents gives companies the opportunity to not only get the architecture right, but also make the right investments in their systems to allow for the sort of governance, quality, and management they probably should have had in the first place.

So, what should you look for in a SOA quality tool? To sum up, here are the key capabilities:

- ***Full lifecycle, continuous testing*** – in the SOA context, quality is a never-ending battle, not just one phase in your development process.
- ***Focus on quality above and below the Service abstraction*** – Services abstract running software. If the software isn't working, then the Services aren't either.
- ***Support for collaboration across diverse teams*** – successful SOA initiatives involve several participants across the organization. SOA quality touches all of them.
- ***Scope well beyond Web Services*** – Web Services have an important role in SOA, to be sure, but they are neither necessary nor sufficient. Your SOA quality tooling cannot depend on Web Services testing.

If you haven't built quality into your architecture, then it doesn't matter how sophisticated your SOA testing tools are.

If you haven't built quality into your architecture, then it doesn't matter how sophisticated your SOA testing tools are. As with so many other aspects of SOA, the tools don't give you the best practices. Instead, the best practices of SOA help you get the most out of your tools. Organizations who get such best practices wrong often simply have to rework their SOA and start again, chalking their early adopter efforts up to experience. That's fine for those early adopters who were blazing the SOA best practices trail. Organizations who are only now framing their SOA plans for the first time, however, have no excuse for not building quality SOA the first time.

Copyright, Trademark Notice, and Statement of Opinion

About ZapThink, LLC

ZapThink is an IT advisory and analysis firm that provides trusted advice and critical insight into the architectural and organizational changes brought about by the movement to XML, Web Services, and Service Orientation. We provide our three target audiences of IT vendors, service providers and end-users a clear roadmap for standards-based, loosely coupled distributed computing – a vision of IT meeting the needs of the agile business.

ZapThink helps its customers in three ways: by helping companies understand IT products and services in the context of Service-Oriented Architecture (SOA) and the vision of Service Orientation, by providing guidance into emerging best practices for Web Services and SOA adoption, and by bringing together all our audiences into a network that provides business value and expertise to each member of the network.

ZapThink provides market intelligence to IT vendors and professional services firms that offer XML and Web Services-based products and services in order to help them understand their competitive landscape, plan their product roadmaps, and communicate their value proposition to their customers within the context of Service Orientation.

ZapThink provides guidance and expertise to professional services firms to help them grow and innovate their services as well as promote their capabilities to end-users and vendors looking to grow their businesses.

ZapThink also provides implementation intelligence to IT users who are seeking guidance and clarity into the best practices for planning and implementing SOA, including how to assemble the available products and services into a coherent plan.

ZapThink's senior analysts are widely regarded as the "go to analysts" for XML, Web Services, and SOA by vendors, end-users, and the press. Respected for their candid, insightful opinions, they are in great demand as speakers, and have presented at conferences and industry events around the world. They are among the most quoted industry analysts in the IT industry. ZapThink was founded in November 2000 and is headquartered in Baltimore, Maryland.

ZAPTHINK CONTACT:
ZapThink, LLC
108 Woodlawn Road
Baltimore, MD 21210
Phone: +1 (781) 207 0203
Fax: +1 (815) 301 3171
info@zapthink.com

ZapThink, LLC • 108 Woodlawn Road • Baltimore, MD 21210 • info@zapthink.com • www.zapthink.com