The “Three C’s” of SOA testing
Software Quality Strategies for Dynamic Apps and Distributed Teams

John Michelsen
Chief Architect, iTKO

iTKO White Paper
July 2006
Executive Summary

Services-Oriented Architectures, or SOA, have allowed companies to flexibly string together new and existing software components as services in a business workflow. These next-generation apps are no longer built through long-term “big-bang” projects. Instead, they are integrated as workflows in smaller intervals, and consumed at runtime.

Services-based business applications are entering the mainstream. Gartner estimates that 60% of enterprise development groups are planning or in the midst of an SOA initiative for 2006, and by 2009, more than 80% of development and integration budgets will be dedicated to applications delivered as an SOA.

Quality will become a primary governor on the enterprise’s success in successfully achieving the promise of SOA. With so many interconnected parts making up applications that can be delivered virtually anywhere, testing no longer becomes a mere matter of finding bugs within the developer’s code, or problems that occur on the screen a given user interface. Software quality processes must evolve with the architecture, to genuinely test a business process and maintain context across the entire workflow.

SOA development and deployment is rendering traditional ways of testing obsolete. There are three primary software quality differences that are introduced by SOA:

1. SOA is comprised of heterogeneous technology
2. SOA teams are distributed, making accountability for quality difficult to achieve
3. SOA architectures are dynamic, interdependent and continuously evolving

Enterprises can improve SOA quality levels by achieving the three C’s of SOA testing:

1. **Complete** testing of business workflows across every heterogeneous technology layer of the SOA, at both a system and component level.
2. **Collaborative** involvement of the whole team in quality. Enable both developers and non-programmers to define and share test cases that prove how the SOA meets business requirements during the entire application lifecycle.
3. **Continuous** validation of a changing SOA deployment as it is consumed at runtime, ensuring that business requirements will continue to be met as the system dynamically changes.

**ITKO LISA 3 Complete SOA Testing System** was built from the ground up to offer complete, collaborative and continuous testing for the next generation of SOA applications. This paper will discuss how your team, and all your business partners and constituents, can finally achieve enterprise-strength quality in an SOA world.
Current State of SOA Strategies

Definition of SOA
What is SOA? There are some misconceptions in the marketplace about what makes up an SOA strategy. Some think SOA is merely a substitute term for web services or SOAP/WSDL. Other companies look at SOA as a “kitchen sink” approach to pulling together several legacy applications and stitching them together using various adapters – basically another way to describe integration. Both of these concepts are part of SOA, but ultimately fail to provide an appropriate definition.

In reality, SOA, or “Services-Oriented Architecture,” is a broad business strategy which encompasses the entire application delivery and deployment lifecycle, and it is not simply about web services or integration of legacy apps. With an SOA, a business workflow may span multiple types of technologies and components that are exposed to other systems as services. These services do not have to be created or hosted by any single department or company. But don’t let the “alphabet soup” of technology acronyms inherent in SOA daunt your team’s progress toward more responsive systems. The benefits of SOA thinking are within reach of most companies.

What is the promise of SOA?
SOA has become more than a preferred approach to application delivery – it has become a competitive mandate for any enterprise that relies upon software for core business processes. There are four primary goals driving the adoption of SOA:

- **Reduce integration cost**
  - “Loosely-coupled” forms of integration allow a software component to be created and used independently and stitched together with other components to create a workflow.
  - Industry standards for integration, such as the WSDL specification for web services definitions, provide some common language for communication and integration between developed components, whether they reside within the same development group or within other departments and companies.

- **Increase asset reuse**
  - A service component within an SOA can be leveraged within multiple applications or workflows. The service may be consumed by other applications or components, and it may consume or rely on data from other services.
  - Legacy databases and application components can be exposed as services within an SOA, meaning the business will be able to maintain key existing business logic where it resides, rather than rebuilding it from scratch.

- **Increase business agility**
  - The ability to rapidly leverage existing components to be responsive to customers and provide new business capabilities is a primary motivator for adopting SOA.
  - With the ability to assemble components, businesses gain better control over their workflows and can quickly adjust them based on customer needs.

- **Reduce business risk**
  - Application failures will continue to cost businesses billions of dollars in lost profitability. With an SOA, companies can simply repair or replace smaller units of functionality, mitigating some costly failures by containing them to smaller problem areas that are repaired quicker.
Executives are increasingly held accountable for failure to provide accurate financial reports and forecasts. Governance, compliance efforts, and risk reduction are enabled through increased business visibility and agility.

According to a December 2005 Aberdeen Group study, a $10 billion company with a $300 million IT budget can save $30 million per year from a broad SOA adoption after a five-year horizon of implementing SOA in at least 75 percent of its applications. But before you can count these savings, you have to ensure that the SOA implementations will successfully meet business requirements. Quality is a mandate for success in SOA, even more so than in past architectures.

**Challenges of SOA testing**

Why is SOA testing such a different beast than previous forms of browser, client/server and mainframe testing? Many of the benefits of SOA become challenges to testing an SOA application.

1. **SOA is comprised of heterogeneous technology**
   
   SOA systems are based on heterogeneous, distributed technologies. No longer can we expect to test an application that was developed in a standard language, by a unified group, as a single project, sitting on a single application server and delivered through a standardized browser interface. The ability to string together multiple types of components to form a business workflow allows for unconstrained thinking from an architect’s perspective, and paranoia from a tester’s perspective.

   In SOA, application logic is generally not housed at the user interface and database level as in client/server, nor is it exclusively stored in a back-end as in venerable mainframe systems. The business rules and behaviors of an SOA live in the middle tier, operating within any number of technologies, residing outside the department, or even outside the company.
The post-SOA world, as shown here, offers a vast array of options for how you assemble or consume business workflows across multiple technologies, both inside and outside your business infrastructure.

In a distributed, heterogeneous application environment, new issues and errors arise when you make multiple components talk to each other to support a business requirement. In an SOA, more unique technology types, multiplied by more points of connection = an exponential increase in possible failure points.

2. SOA teams are distributed and lack accountability

Ensuring quality within a single development team, working on a single complete implementation, was difficult enough. Now, like it or not, you have a much larger, highly distributed team to match the heterogeneous nature of the SOA. With an SOA, application “stress points” can be anywhere, and will change as multiple parties inside and outside of your team can add or modify the individual services your application relies on or contributes to.

There is a quality chasm between Unit and Acceptance Testing. Finding the root causes of problems across the middle tiers of SOA applications is difficult. Testing a front end user interface becomes irrelevant when it provides no insight into what is actually happening on the back end. And expecting developers to find missed requirements by conducting more unit testing at the code level doesn’t get the team there either — it may find some bugs in the component-level code, but it won’t demonstrate why a business requirement isn’t being met.

Bridging the “Quality Gap”

In an SOA world, testing can no longer remain siloed to specific roles and by phases to development or QA teams. The entire extended organization of partners, stakeholders and constituents must collaborate on testing, because distributed development and QA teams make accountability hard to enforce.

3. SOA architectures are dynamic, interdependent and continuously evolving

The services architectures of today are bound to change with the enterprise and with technology in general. Complexity is driven into software by the natural process of competition, which forces the rapid evolution of new business rules and logic into business systems. According to the 2005
Aberdeen report, “It’s no surprise that the top factor for implementing SOA, which 50% of survey respondents cited, is development of new capabilities.” Business competition will continually force SOA evolution, whether or not your systems have been validated under the new business conditions.

**Testing SOA is less about structure, more about flow**
Selecting an SOA approach is not like designing and building a house for your applications. It is an ongoing process for moving forward into a world of options. No company can afford to take a 1-3 year break from business while the systems are rebuilt. Your apps must constantly integrate with new and existing tools, partners and legacy apps, while gradually moving toward the ideal strategy. And unlike the old client-server days, this test phase does not end at deployment.

Consider a typical SOA implementation found within a larger corporation. The company will have several internal and end-user applications it must build and support. To deploy these apps more rapidly at lower cost, the applications can share and consume data and business logic housed within legacy apps and other partner services through an integration layer (or Enterprise Services Bus in some companies). The apps, services, and enabling business rules can all be shared through one integration layer. But what is happening inside the pipe as services change, and interactions between them changes as well?

In an SOA, testing as a phase that occurs during component development and integration is useful, but it won’t help you trap for unintended consequences. Since SOAs involve many moving and evolving parts that are constantly leveraging each other, your testing capabilities must evolve from “What went wrong?” to “What if?,” so you can continuously, proactively test for current and future issues.

![Integration Layer (ESB)](image)

**Figure 4. Unintended Consequences** occur when you try to correct one aspect of an SOA application without visibility into the interdependent services it interacts with and the other applications that consume those services. Making a change to one service may fix the root cause of your application error, while potentially disrupting any other element of the SOA.

How can you consistently test, when you are trying to hit a moving target with fragile manual tests? **Change is constant in SOA even after deployment, so you must be able to continuously monitor for unintended consequences.**
Solution: Complete, Collaborative, Continuous Testing

Getting the whole team to overcome chaos and own quality is essential, especially in an SOA, where the rate of change is extremely high. As processes are rolled out faster, with higher degrees of flexibility, testing must keep pace with constant change to ensure business viability.

Successfully achieving SOA quality requires three key abilities:

1. **Complete** testing of business workflows across every heterogeneous technology layer of the SOA, at both a system and component level.
2. **Collaborative** involvement of the whole team in quality. Enable both developers and non-programmers to define and share test cases that prove how the SOA meets business requirements during the entire application lifecycle.
3. **Continuous** validation of a changing SOA deployment as it is consumed at runtime, ensuring that business requirements will continue to be met as the system dynamically changes.

How the above goals are met of course depends on the structure and responsibilities of your organization. There are both process and technology enablers for each, which you can implement for starters in a specifically “agile” extended team.

1. **Complete testing across every heterogeneous technology**

Composite SOA applications are no longer hosted within a single business or created by a single development group. These new types of software consist of multiple layers of different technologies, tied together by protocols and messaging standards over the Internet. The increase in flexibility gained by being able to integrate so many components is offset by an increase in the potential points of failure where these systems connect.

You need to be able to test a business workflow across the alphabet soup of SOA:

- Web interface (HTML, .ASP, JSP, PHP, HTTP, JavaScript, AJAX, etc.)
- Web Services (Java or .NET, SOAP, WSDL, XML, UDDI)
- Java servers (J2EE, EJBs)
- Other program objects (RMI, POJO, Swing fat clients, etc.)
- Databases (JDBC, SQL, etc.)
- Messaging layers (JMS/MQ)
- File systems
- And extend testing to MANY other flavors of custom or legacy objects.

Additional Unit Testing or Code Coverage is useful for finding structural bugs within a component, but not for validating that a component or application is meeting business requirements as it
interacts with the real world. Missed business requirements or misunderstandings are generally the costliest mistakes, as the cost of automating unit test execution tends to be lower.

More functional testing of the finished UI won’t solve the quality problem for most implementations – even when bugs are squashed out of each component, new issues arise when you try to get these components to talk to each other to form a business workflow. An Agile Quality Organization will allow the SOA to be completely tested, both at a component level and an integration or system level.

2. Collaborative testing: Make everyone agile.
By now, anyone involved in business IT is somewhat familiar with the concept of Agile programming: shorter iterations of a smaller feature set, continuous releases, and continuous testing. Also known as “test first” or “XP” methods, Agile is essentially about making your development process more responsive to the changing needs of your business. Companies with agile development practices can often outmaneuver competitors by reacting faster to business changes and customer demands for specific functionality.

Agile is first and foremost about testing. Without effective tests that accompany each iteration, you cannot confidently introduce a feature that will work for the business. So in Agile development environments, quality is the governor on the velocity of your entire IT infrastructure.

The simple fact is, developers are usually the least qualified to test their own applications. It’s not that they aren’t smart enough to find bugs. Developers like to build software, not break it. Expecting developers alone to understand and test for business requirements when they are occupied in the throes of production is plain ridiculous. However, business stakeholders and QA teams with a deeper understanding of business requirements are not usually technical-minded enough to code tests, so there is no way for them to contribute until the iteration is near delivery.

Agile without Collaboration: for Developers Only?

Here’s where most agile processes fail – they are oriented toward coding tasks and unit testing alone. Therefore only developers are able to get involved at present, due to the high level of technical expertise required to write a new program to test. When only developers are testing, QA and business analysts become a blockade to completing each iteration and the process reverts to a waterfall “test at the end” approach.

Tightening Dev/QA/Business teams to collaboratively test makes your team highly accountable and agile by comparison. iTKO has always recommended this type of “pair testing” approach.
Distributed Technologies = Distributed Teams Must Own Quality

The distributed nature of services accelerates the development of composite SOA workflows. However, this means quality is no longer a matter of a single department or team. Many components of SOA applications can be developed and maintained by other departmental teams, or other business partners and companies – each on their own development or release cycle.

If your SOA application is mission critical, a best practice is to bring your most important development partners into the accountability fold, so you can get better visibility into changes and issues across the most important functions your app must support. In addition, you can also test the interactions of your systems with partner systems on an ongoing basis to ensure accountability.

Agile when Everyone in the Distributed Team Owns Quality

As in the above example, everyone should be involved in creating and executing tests as code and components are introduced. An agile organization allows both developers and non-developers, in multiple teams, to collaborate on testing components of the application long before the end of development. This ensures continuous testing at each iteration cycle and a more reliable course correction and release process.

3. Composite Apps Need Continuous Testing

In an environment of heterogeneous components and distributed teams, we need a way to tie these elements together so the business requirements are actually met by SOA.

Since SOAs are highly interdependent workflows that are flexibly changing and consumed at runtime, you need continuous testing before and during deployment. The components of SOA may be tested to perfection in isolation, but they must work together to form composite apps. Any service you leverage may have several upstream services that feed it information, or downstream services that consume its output. You can build a workflow out of as many services as you like. So how will you know that they are working together as expected, if change is inevitable?

For instance, let’s say your team is responsible for quoting the prices for an airlines reservation system. Your app calculates current available seats and rates by querying upstream services created within your group (for seat availability and expected reservation rates), external services (for pricing and price comparison), and other sources of independently managed data like flight
In a services world, what happens when one of your travel portals starts reporting false data on seat availability? You can try root cause analysis on the problem and perhaps find the perceived source of the problem in a service or database. But what will happen to the rest of the connected services and apps of the SOA if you make a change?

Continuous testing gives you the visibility to know "what if," not just "what is." When you can constantly run tests against all connected elements of the SOA, you gain a high degree of transparency into what will happen when interdependent services are modified.

Testing should be able to point out unintended consequences of any change at all phases of development, deployment and support of an SOA application. When releases are continuous, there is no longer any reason to treat any phase of testing as a set event handled as a separate discipline.

How LISA Achieves the 3 C’s

iTKO LISA 3 is the first solution built from the ground up to answer the three C’s and overcome the testability chaos inherent in SOA environments.

How LISA meets the requirements of SOA Testing

<table>
<thead>
<tr>
<th>Challenge</th>
<th>With LISA</th>
<th>Other Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete testing of heterogeneous services</td>
<td>LISA can test every service technology with one tool</td>
<td>Multiple unintegrated, separately developed tools that test some point technologies</td>
</tr>
<tr>
<td></td>
<td>LISA can test each tier of an application, regardless of location</td>
<td>Usually web-only or Web Services only, applicable for internally developed/hosted components</td>
</tr>
<tr>
<td></td>
<td>LISA shines at testing applications before the UI even exists, as well as providing testing for multiple UI types</td>
<td>You must build a test client UI for middle tiers before testing and/or rescript all tests against changing dynamic UIs</td>
</tr>
<tr>
<td></td>
<td>LISA was built from the ground up</td>
<td>Separate testing tools per</td>
</tr>
<tr>
<td>Collaborative testing for the entire extended team</td>
<td>LISA works seamlessly with SOA security and service registration protocols</td>
<td>Difficult to dynamically maintain sessions and identity data, requires manual intervention or adaptation</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Continuous testing of interdependent SOA applications</td>
<td>LISA highlights stress points AND business logic errors by including functional tests in performance monitoring</td>
<td>Performance testing and monitoring are largely statistical – conducted in total isolation from functional validation and root-cause analysis</td>
</tr>
<tr>
<td></td>
<td>LISA will report root cause info within the UI and components, and supports most application lifecycle management processes</td>
<td>Primarily tests the user interface and screen-level interaction, test management is either a separate discipline or you must use the test vendor's process to share tests</td>
</tr>
<tr>
<td></td>
<td>LISA is a distributed application – put it where your application resides, and wire results securely to your support organization</td>
<td>Requires integration and licensing of a test engine per target server, difficult when services are owned/managed by any group</td>
</tr>
<tr>
<td></td>
<td>LISA tests grow and evolve with the business logic</td>
<td>Tests require a near completed state application to validate business logic, and tests are only applicable to that version</td>
</tr>
<tr>
<td></td>
<td>The LISA Server can stage and schedule full functional and load tests to run at any time interval or launch from any build or event</td>
<td>Load and performance testing can run at scheduled times, but not easy to integrate with development events and provide little functional validation of the SOA app, beyond statistical or “uptime”</td>
</tr>
<tr>
<td></td>
<td>LISA Extensibility allows all SOA components to be “test enabled” with a simple API to self-report to a continuously running instance of LISA, which can send alerts or note exceptions</td>
<td>Custom integration only on a per-component basis, little means to connect the functionality of components to the running test</td>
</tr>
</tbody>
</table>

iTKO LISA enables SOA Quality Maturity for the Organization

Many organizations are actively trying to increase their overall reliability to meet IT requirements on time and budget by evolving their quality process and testing methodologies.

We can compare the maturity of SOA software testing to the overall software development maturity model espoused by other industry standards such as TQM, ISO9001, Six Sigma and the like, for ensuring that a business supplier is maintaining high levels of process control and efficiency.

To achieve higher levels in quality maturity in the above matrix, a company would need to be automating the creation and staging of that respective type of testing in a repeatable, meaningful way along the “Three C’s” of SOA Quality. Don't settle for anything less.
**Conclusion**

**iTKO LISA 3** is the first complete SOA software automated testing platform. SOA has become the preferred strategy for delivering critical enterprise applications, but software testing has not kept up with the breadth, scope and depth of testing requirements of SOA until now.

Quality is at the core of responsible enterprise development. You can accelerate development and reduce costs through better application lifecycle management (ALM) practices. But fundamentally, you cannot deliver a piece of business software unless you can ensure that it meets the business requirements and will not fail in front of the end customer. Therefore, quality has become a governing factor on the effectiveness and value of enterprise software.

**iTKO LISA helps companies attain higher overall SOA quality levels and faster development by enabling the three C’s of SOA testing:**

1. **Complete** testing of business workflows across every heterogeneous technology layer of the SOA, at both a system and component level.
2. **Collaborative** involvement of the whole team in quality. Enable both developers and non-programmers to define and share test cases that prove how the SOA meets business requirements during the entire application lifecycle.
3. **Continuous** validation of a changing SOA deployment as it is consumed at runtime, ensuring that business requirements will continue to be met as the system dynamically changes.

You want to achieve the value you expected from an SOA approach to enterprise development, but in the past, genuine automation of test creation and execution was infeasible. Ask yourself:

- What is the cost of building and maintaining each test and how do tests contribute to the overall SOA development strategy?
- And, can my testing actually ensure I am meeting the business requirements in today’s SOA environment?

iTKO’s mission is for **everyone to own quality**. iTKO LISA 3 helps enterprises achieve maturity in software quality, enabling complete, collaborative and continuous SOA testing at every technology layer and at the business process level. Only LISA offers a single, comprehensive quality platform that maintains business context to test all of the distributed layers that make up today’s SOA software.