SOA application testing
White paper
May 2007

Testing SOA applications with IBM Rational quality management solutions.

Contents

Introduction

Service-oriented architecture (SOA) makes IT applications into composite applications, which are no longer monolithic. Instead, composite applications are composed of many services often developed and deployed independently by separate development teams on different schedules. Development of new composite applications is made easier by the possibility of reusing existing services, thereby avoiding costly application redevelopment or integration. However, this comes with some unique challenges to ensuring a high level of quality throughout the development cycle.

Indeed, SOA quality management is an important aspect of service lifecycle management—one that reflects the need to address multiple aspects of service quality across multiple SOA service implementations. IBM is focused on delivering end-to-end SOA quality management—from the model phase through the assemble, deploy and manage phases. SOA quality management concerns far more than just conventional software development and testing. It encompasses all the ways in which business and IT organizations collaborate on services, as well as the lifecycle from the conception of services and composite business applications to the retirement of those assets.

The key capabilities that IBM SOA quality management solutions deliver include:

- *Enabling through tools and best practices a quality management focus throughout the SOA lifecycle.*
- *Ensuring business agility by enabling the functional and performance testing of business services for compliance with business and regulatory requirements.*
- *Optimizing and automating workflows across business processes by streamlining processes and eliminating process redundancies.*

The SOA quality management processes within each service lifecycle management phase include the following activities:

Model

- *Validate your business requirements.*
- *Discover and assess those requirements against your current services.*
- *Model your service requirements.*

Assemble

- *Create your service update plan.*
- *Create or modify the services to meet the business requirements.*
- *Assess the services against your governance rules.*

Deploy

- *Test the quality of the services.*
  - *Function testing*
  - *Performance testing*
  - *Compliance testing*
- *Approve the deployment of the services.*

Manage

- *Manage and monitor the services throughout their lifecycle.*
- *Track the services in the registry.*
- *Report on the services against service-level agreements.*

SOA characteristics and testing challenges

Service-oriented architecture is a business-centric IT architectural approach that supports integrating your business as linked, repeatable business tasks, or services. SOA composite applications are built by assembling standardized components (services) that are reused and combined to address changing business goals and priorities. This is true business driven development where business analysts document and optimize business requirements in the form of business process models that spread across multiple applications. Therefore, effective and comprehensive quality management calls for understanding the business processes across several composite applications.

Moreover, SOA focuses on business flexibility—the capability to deploy innovative business models quickly with reusable and optimized processes. Ensuring a constant level of high quality in such an ever-changing environment requires agile and continuous testing across the software delivery lifecycle, and those activities need to be planned from the very beginning of a new SOA implementation project.

Another key element of the SOA approach is to build loosely coupled business processes and services. This allows you to meet the objectives of business flexibility and reuse—meaning that services can and will be reused in applications and contexts that are not known at the time they are built and tested. The lack of well-defined boundaries and control over the entire IT solution under test creates new challenges in the definition of an appropriate and effective test strategy. The SOA defines several layers of abstraction with the goal of decoupling software components that were previously tightly integrated in order to enable agility and reuse across the board. This is realized by the services layer that defines abstract and GUI-less entities that map to the repeatable business tasks of an organization, and that hide the underlying software components.

In that architecture, services do not usually provide human-friendly interfaces that we traditionally use to perform functional validation either manually or in an automated manner. Instead, testers have to deal with messages and protocols based on technology such as the Web services standards. And, services expose coarse-grained interfaces that hide all the details of the underlying business logic. They appear as a "black box" to the service consumers. It becomes challenging for quality engineers to find the right inputs in order to perform exhaustive test coverage of those services.

Reuse is another benefit of adopting a service-oriented architecture. Because an SOA allows you to avoid building point-to-point connections between every application, you can reduce both your development and maintenance costs. One single service can now be used by several applications, or maybe a few at the beginning of the adoption of SOA, but soon, it could happen that some services will be consumed by a dozen or more applications. If those services have not been designed and tested to perform under such load, some of the applications using them might suffer from unacceptable response times. And, in the worst case—where services have not been thoroughly tested and fail in production—the impact of that failure can be catastrophic for the whole organization, as several applications would be affected at the same time.

Approach to testing SOA applications

We have just seen that service-oriented architecture brings many benefits, but it also brings unique testing challenges for the quality assurance (QA) team. So testers should be well prepared and adopt best practices to build an approach that will help them effectively address those challenges. Lessons learned from several large-scale SOA implementation projects have been leveraged to define key elements that can make a QA team successful with testing SOA applications.

First, from the very beginning of a new SOA project, it is very important to put together a collaborative cross-functional test team that understands the business tasks and processes to be tested; the different aspects of service-level agreements; and the underlying technologies being used. In that team, business analysts should be providing clear inputs with regard to test objectives, and they should review and approve the proposed test strategy and test plans. Test architects are the main stakeholders of the test plans being defined by the team. They should take into account the service-level agreements that describe the expected level of quality of service, performance, fault tolerance, etc. Testers, performance engineers and developers should be trained on the SOA technologies and tools, and they should be ready to effectively turn test plans into executable test cases.

This collaborative test team, once put in place, should rely on the well-known mantra "test early, test often." It is far more effective to find and fix defects close to where they were introduced than it is to find them later on, when it becomes harder to spot the cause of problems—and when fixing them has a much greater impact on the application code and design. In a service-oriented architecture, where there is less control over the entire solution being built, and where services will be reused in different applications and in many different ways, it is recommended to test each service individually first. For example, will services that are reused still perform well with an additional load? Validating the functions and the performance of services being built, as well as the ones being reused, would reduce the number of issues found during the integration phase.

The composing of services is usually done by designing the workflow using business process models, most of the time using the state machine paradigm. Sometimes, these models are turned into executable code by using the Business Process Execution Language (BPEL). In all cases, understanding those interaction models is helpful in creating integration test scenarios that involve the right services together, with the correct combination.

Of course, the test team will test the complete solution. But it should also validate that the new application meets all the business objectives for which it is being implemented. Obviously, there is no benefit for an organization to adopt a brand-new, bug-free, well-performing application that no one will actually use because it does not meet one of the business objectives. With an SOA approach, testing, like all the other development activities, should be business driven.

Certainly, continuous testing along the entire software delivery cycle is resource intensive—unless you automate the creation and execution of most of the tests. Effective test automation for SOA applications is a challenge in itself that calls for a rigorous approach. Test automation should be considered a task on its own, and not just an activity that testers perform in their spare time or after the fact, when everyone realizes the project is behind schedule. Instead, a test automation project needs to be run just like any software development project: with a detailed plan that includes requirements, objectives and resource allocation.

One of the problems that often occurs with test automation is the application user interface changes frequently during development—sometimes from one build to another. This means that automated test cases are broken and need to be updated frequently. In an SOA, if most of the tests have been created and automated based on the service layer, and therefore are independent of the user interface changes, this will offer better maintainability and will save cost and time.

Another way to optimize the test automation activity is to reuse test assets as much as possible. For example, a good practice is to create libraries of executable test steps that can be put together to build more complex test cases. A data-driven testing approach could also be applied where the same test scenario is reused several times with different sets of data.

Moreover, effective test automation calls for tools that can ease the creation and maintenance of both functional and nonfunctional tests in relation to the development platform used on a project, specifically with the change and test management tools. And, in the context of a service-oriented architecture, the test tools should address the test of GUI-less services and support the underlying open standards and technologies.

Rational software solutions for testing SOA applications

As part of its new SOA quality management solution, IBM has announced two new solutions from Rational software to enable continuous testing of services and SOA applications.

- ***IBM Rational® Tester for SOA Quality*** *software is for developer and quality assurance professionals who need to create, comprehend, modify and execute both functional and regression tests of GUI-less services.*
- ***IBM Rational Performance Tester Extension for SOA Quality*** *software is for performance engineers who need to conduct load and performance testing of services. It extends IBM Rational Tester for SOA Quality and IBM Rational Performance Tester software.*

Rational Tester for SOA Quality simplifies the creation of tests for GUI-less services by automating the generation of Web service test client or by recording the interaction between a service consumer and a service provider. It comes with a graphical test editor that enables both high-level and deeper detail views. No programming knowledge is necessary to create, modify and execute functional tests. Moreover, the software leverages the business modeling work developers have already done, and it automatically generates test cases based on the behavior of business processes defined in BPEL. Rational Tester for SOA Quality can automatically detect variable data during test recording and prepare the test for data-driven testing. And it allows users to easily create customized data sets to be used by the tests during execution.

Rational Performance Tester Extension for SOA Quality provides a flexible workload modeling capability, enabling automated generation of service performance tests. This modeling helps ensure that the performance test accurately mirrors the user base, including different groups of service consumers, as well as the activities and usage patterns of each of the groups. Through the modeling exercises, this tool can provide more effective real-world scenarios.

Now you can easily pinpoint the performance bottlenecks of your SOA application. To help find the root cause of the bottlenecks, Rational Performance Tester Extension for SOA Quality allows you to import IBM Tivoli® Monitoring response time breakdown data.

Let's get into more details of how these tools can be used. Rational Tester for SOA Quality supports three different ways to create a new test. First, interactively given the description of a service as input, the tool automatically creates Web pages that allow the tester to easily interact with the services to be tested. Second, the tool can record the traffic between an existing service consumer and a server, either by an HTTP proxy or by instrumentation technique. Third, the software allows the tester to describe a business process in BPEL, which will be analyzed to automatically generate tests that will exercise the different possible paths through the business process.

During the creation of a test, interactively or by recording, the tool automatically detects variable data, such as unique session or transaction ID, and if necessary, automatically correlates field values between several service operation invocations. This allows testers to automatically execute the recorded test without having to manually change to it.

Once tests have been recorded, users will be able to easily enhance or modify them using a visual editor. Tests are represented in a tree view showing the list of operation calls and responses. With the editor, users can modify or add new operation invocations, and also interactively call any one of the listed operations to update the recorded answer. Test data that is constituting the Simple Object Access Protocol (SOAP) messages to be sent to call a service operation can be displayed and edited through multiple views, which either focus only on the

values or on the display of all the elements including name spaces, attributes or attachments. Finally, the editor provides easy access to the configuration of the transport protocol and Web Services Security that are used to execute the tests.

One way to enhance tests is to use a set of data instead of just single values to execute the same tests several times in a test-data-driven approach. And, if needed, users can easily add correlations between message fields on top of the one automatically created by the tool during the recording.

Another important aspect of this step is to set the baseline that will be used to automatically determine the verdict of a test execution. This is done by creating verification points from the visual editor. Verification points will, for example, automatically compare an answer from a service call with the expected answer. Once the test has been recorded and enhanced, it is ready to be run. This can be done directly through the tool user interface, but also from the command line, or from the IBM Rational ClearQuest® test management solution. So, you can easily create repeatable test suites that can be run as part of an automatic build process.

As the result of the test execution, the tool provides a functional test report with the global verdict of the executed tests—pass or fail—and an execution history view with all the messages exchanged. The test log also includes the verdicts for each verification point, as well as a detailed view showing a side-by-side comparison between the expected and received responses of the operation invocations.

In addition to validating the functional requirements of services, Rational Performance Tester Extension for SOA Quality allows performance engineers to easily create and execute load tests. A visual schedule editor is used to define powerful and flexible scheduling of tests in order to accurately model production-like workloads with hundreds or thousands of service consumers executed in parallel.

You can create specific tests to be scheduled, or reuse the function tests already created for the validation of the services, thereby cutting down on the cost and time needed to benchmark your SOA applications. The load generation can be distributed on several machines, including IBM mainframes, to mimic closely the production environment where the services and applications will be deployed.

Just like the functional tests, the performance tests can be executed from the tool's user interface, the command line or the Rational software test management solution. During the test execution, detailed response time data is presented in customizable charts and tables, which can be exported into HTML reports. At the same time, information relative to the utilization of system resources can be gathered on both the load generation machines and the systems under test, such as CPU utilization or memory consumption.

On top of that, the tool allows you to collect more detailed server-side data leveraging the Tivoli monitoring infrastructure, such as the breakdown of response time for each tier of the system. This enables performance engineers to quickly identify the root cause of performance problems.

Summary

Knowing which new challenges that key SOA characteristics bring to testing activities is helpful for you and your team. Some of these challenges will require adjustments to your current quality management and testing processes, and they will demand that you implement new tools specifically designed to deal with new assets under test. IBM Rational Tester for SOA Quality and IBM Rational Performance Tester Extension for SOA Quality address those needs and are effective solutions to tackle the challenging tasks of testing service-oriented applications.

For more information

To learn more about IBM Rational Tester for SOA Quality system requirements, visit:

ibm.com/software/awdtools/tester/soa

To learn more about IBM Rational Performance Tester Extension for SOA Quality, visit:

ibm.com/software/awdtools/tester/performance/ext/soa

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-07

RAD10993-USEN-00