Software Engineering Institute

Testing in Service-Oriented Environments

Ed Morris
William Anderson
Sriram Bala
David Carney
John Morley
Patrick Place
Soumya Simanta

March 2010

TECHNICAL REPORT
CMU/SEI-2010-TR-011
ESC-TR-2010-011

Research, Technology, and System Solutions (RTSS) Program

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Table of Contents

List of Figures

List of Tables

Abstract

This report makes recommendations for testing service-oriented architecture (SOA) implementations consisting of infrastructure, services, and end-to-end processes. Testing implementations of SOA infrastructure, services, and end-to-end processing in support of business processes is complex. SOA infrastructure is often composed of multiple, independently constructed commercial products—often from different vendors—that must be carefully configured to interact in an appropriate manner. Services are loosely coupled components which are intended to make minimal assumptions about where, why, and under what environmental conditions they are invoked. Business processes link together multiple services and other systems in support of specific tasks. These services and systems may operate on remote platforms controlled by different organizations and with different SOA infrastructures. Such complications make it difficult to establish appropriate environments for tests, to ensure specific qualities of service, and to keep testing up-to-date with changing configurations of platforms, infrastructure, services, and other components.

1 Introduction

Service-oriented architecture (SOA) has quickly become a dominant paradigm for distributed, interoperating, software systems of systems.[1] Three interrelated concepts need to be distinguished: SOA, services, and business processes.

1. SOA is a way of designing, developing, deploying, and managing systems that is characterized by
   a. coarse-grained services that represent reusable business functionality
   b. service consumers that are clients for the functionality provided by the services, such as end-user applications, internal and external systems, portals, or even other services (i.e., composite services) and that compose applications or systems using the functionality provided by these services through standard interfaces
   c. an SOA infrastructure that provides means to connect service consumers to services [1].
2. Services are reusable components that represent business tasks, such as customer lookup, weather, account lookup, or credit card validation. Services represent a task or step in a business process and they can be globally distributed across organizations and reconfigured to support new business processes.
3. Business processes (also referred to as mission threads by some government organizations) are a defined set of tasks that produce specific capabilities or products. Each task is composed of one or more services that are integrated to form the process.

While there are many potential implementations of the SOA architectural style, web services that include HTTP, SOAP, WSDL, and UDDI provide one of the most widely adopted approaches to SOA implementation [2,3,4,5].[2] Because web service implementations are so widespread, commercial tool support is available for many aspects of the engineering process, including testing. SOA implementations are possible using other technologies and standards; however, commercial tool support is less widely available for these other implementations.

Whichever technologies and standards SOA implementations use, it is challenging to test those implementations, for various reasons. For instance, services are often outside the control of the organization consuming the service (i.e., service builders, service owners, and service consumers may each be from different organizations), leading to potential mismatches and misunderstandings between parties. In addition, SOA implementations are often highly dynamic, with frequently changing services and service consumers, and varying load on services, SOA infrastructure, and the underlying network. Consequently, it is normally very difficult to replicate all possible configurations and loads during the testing process.

The scope of an SOA testing effort includes which aspects should be tested for (or against), the elements to be tested, and artifacts produced during the testing effort. Typically, testing hand-

[1] Maier [61] defines a system of systems as one in which there is managerial independence, operational independence, and geographic distribution of the elements; continuing evolution; and emergent behaviors such that the system of systems exhibits behaviors apart from those of individual elements

[2] Acronyms used in this report are identified in Appendix A.

books include strategies for testing the completeness, correctness, and accuracy of requirements, architecture, design, documentation, and many other non-executable artifacts. This report does not address testing of non-executable artifacts. Here, we focus on *dynamic* testing [6], which includes unit, integration, and systems-level testing of executable software (for both functional and non-functional attributes). For information about the broader view of testing, see Kaner, Falk, and Nguyen, and Whittaker [6,7].

1.1 Organization of This Report

Section 2 introduces foundational material about SOA implementation testing. Three aspects to be tested are described: functionality, non-functional attributes, and conformance. In addition, the elements to be tested—SOA infrastructure, web services, and end-to-end threads—are discussed. Finally, key roles such as service developer, service provider, and service consumer are examined.

Section 3 details the challenges faced when testing SOA implementations; Sections 4–8 provide specific guidance for testing the aspects of the implementation. Section 9 discusses other techniques that can be used in conjunction with traditional testing in order to enhance the testing process. Section 10 is a short summary of the report.

Throughout the report, we highlight recommendations for good SOA implementation testing practice; the recommendations are compiled in Appendix B.

2 Testing in a Service Oriented System

At the highest level, testing in an SOA implementation does not differ from testing in traditional systems. Specifically, testing must address

- *functionality*: The functional capabilities must be tested to determine whether requirements and other expectations will be (or are being) met. For example, functional testing in the SOA infrastructure can verify that services are made available correctly (i.e., they are published), that the services can be located, and that connections can be established (called *find and bind*).
- *non-functional attributes*: The non-functional characteristics that are essential for determining fitness for operational use, also called quality attributes, must be tested. Table 1 is a list of some quality attributes [8] considered important, with implications from service-orientation on testing.
- *conformance*: Testing can verify conformance to specific guidelines and standards that the organization has chosen to adopt. For implementations based on web services, there are many standards and guidelines that can be found from the W3C [9] the WS-I [10], and OASIS [11].[3] In addition, many organizations develop standards and guidelines specific to their lines of business.

Table 1: Implications of Service-Orientation on Quality Attributes

Quality Attribute	Implications of Service Orientation on the Quality Attribute
Availability	• A service may not be under the direct control of the service consumer. Therefore, there is no guarantee of availability. • Multiple identical instances of a service may be used to enhance availability, leading to a need to test consistency between the instances. • Diverse instances of a service may be used, complicating testing as each implementation must be tested.
Modifiability	• A service-oriented system is meant to be easy to modify. Further, because its internal implementation (e.g., program language, structure) is hidden from service consumers, a service implementation can be changed as long as the service interface is unchanged. The interface can also be modified as long as prior functionality is maintained (i.e., functionality can be added incrementally). Testing is complicated in cases where the testers do not have access to service implementations. In such a case, testing becomes similar to a COTS (commercial off-the-shelf) environment (i.e., black box with significant trust issues) • A new version of a service may not be backwards compatible with earlier, tested versions of the service.
Interoperability	• Services are meant to be interoperable by publishing their service interfaces and using common protocols, however, there is minimum support for semantic interoperability. • SOA infrastructures can differ in the way they implement features (e.g., security). This makes it difficult to invoke services across differing infrastructures. This problem is sometimes referred to as infrastructure federation.

[3] Acronyms used in this report are identified in Appendix A.

Quality Attribute	Implications of Service Orientation on the Quality Attribute
Performance	If implemented using web services, service orientation has a negative effect on run-time performance (i.e., response time and throughput) primarily because • XML serialization, deserialization, validation, and transformation produce performance overhead. • The presence of meta information makes web services messages more verbose and hence larger in size; therefore, the messages need more bandwidth.
Adaptability	Services are intended to be rapidly adaptable to changing business or mission demands. Because the environment is changing rapidly, testing must balance between the desire for full testing and the need for rapid deployment. Automated test suites and regression testing techniques become critical.
Reliability	While standards are available to help with message-level reliability, service-level reliability is still implementation specific. Tests need to be developed for the failures that derive from the distributed nature of an SOA-based system. For example, testing should analyze the behavior when an invoked service is unavailable.
Security	Service consumers can reuse existing services; however, this advantage often comes at a price. An end-to-end thread can cross trust boundaries. Testing must validate that security is maintained even when information flows across untrusted networks [12].

2.1 Artifacts to be Tested

The functional, non-functional, and conformance characteristics to be tested should be considered in the context of the artifacts of a service-oriented system that are available for testing, specifically the SOA infrastructure, the services, and the service consumers. Note that service consumers may be services that invoke other services, or they may be composite services, or even end-to-end threads performing an enterprise-level capability.

SOA Infrastructure

The SOA infrastructure typically consists of capabilities and services to register and discover services, manage metadata, provide security, and deliver messages. The infrastructure may be composed of custom-developed software, commercial products, or some combination of custom, commercial, and open source capability. An SOA infrastructure stack typically consists of at least the following

- a registry
- an enterprise service bus (ESB)
- application servers
- web servers
- data stores and databases

However, there is wide variation in the SOA infrastructure capabilities provided by commercial vendors, in terms of (1) core elements provided by the infrastructure stack; (2) tools to support developing, integrating, testing, and other software engineering activities; and (3) capabilities, such as a repository or support for security (e.g., services for authentication or authorization of access).

Web Services

Testing of web services includes testing individual services and composites of services. An *individual* web service (also called an *atomic* web service) typically provides coarse-grained, business-level capability. The elements of a web service from a testing perspective are the service interface, service implementation, message format, message payload, and service level agreement (SLA).

Composites involve multiple web services that interact to provide a capability. Web service orchestration and web service choreography are two types of composites [13,14]commonly found in SOA environments.

- Web service orchestration directs the actions of one or more individual web services such that they work together to perform some useful function. Important elements of a web service orchestration are the composition code (e.g., WS-BPEL code), the individual web services, and the infrastructure elements and capabilities used to realize the orchestration.
  - Orchestration (quoting from a *BPTrends* online column) [15]:

    Defines a single master controls of all aspects of a process (top-down approach)

    Supports a graphical view of the sequence

    Easily maps to SOA

    Is usually simpler to start with; but often harder to scale to more complex processes

    Is driven by the graphical sequence model, i.e. function follows form

    Represents the state-of-the-practice, and is supported by the majority of tools
- Web service choreography involves cooperating web service participants, in which services act as peers involved in interactions that may be long-lived and rely on state information to direct behavior (i.e., stateful interactions). The goal of these interactions is common and complementary behavior of the participants that cumulatively accomplishes a business goal.
  - Choreography (quoting from an *BPTrends* online column) [15]:

    The overall process behavior "emerges" from the working of its parts (bottom up). No global perspective is required

    Complex work processes are decomposed into work agendas where each autonomous element controls its own agenda

    Easily maps to event and agent based systems

    Is usually more difficult to start, but often easier to scale to complex processes

    Graphical representations can be derived from the process, i.e. form follows function

    Represents the state-of-the-art, and is gaining support with emerging tools

End-to-End Threads

End-to-end threads are the combination of humans, applications, services, back-end applications, and databases that utilize the SOA and network infrastructure to perform a business task.[4] End-to-end threads include services along with other interacting components (human, functional, infrastructural), along with the operating environment.

[4] Along with end-to-end threads, SOA-based applications are tested in this aspect. For simplicity, we use end-to-end thread (or threads) to stand for both types of composites.

Recommendation for SOA Testing

Rec. 1: Develop an SOA test strategy that accounts for SOA infrastructure elements, web services (individual and composites), and end-to-end threads.

2.2 Perspectives, Roles, and Responsibilities

As indicated in Table 2,[5] there are several roles that organizations and individuals play in developing an SOA implementation. Each role is important in SOA testing, and failure to define the role-specific expectations regarding testing can lead to problems in testing and issue resolution. For example, it is important to determine the role of a COTS vendor in testing and resolving issues with its product in the user's context. Clear expectations can help avoid the finger-pointing that sometimes occurs when products do not interact as expected or promised. Likewise, defining the role of individual service providers in relation to testing of composites (orchestrations and choreographies) or end-to-end threads simplifies the processes of setting up the environment, executing tests, and analyzing results.

Table 2: Important Roles in an SOA Environment

Role	Action	Recommendation for SOA Testing
Service developer	Creates the interface of an individual service and its underlying implementation by using an existing component and wrapping it as a service or forming the service implementation "from scratch"	*Rec. 2: Establish guidelines for testing the service before it is ready for use either by the service consumers or the service integrator.*
Service provider	Provides services. A service provider may or may not be the developer of the service; however, a service developer can also be a service provider.	*Rec. 3: Establish clear lines of testing and customer support responsibility that allow for distinctions between the roles of service developers and service providers.*
Service consumer	Uses a service (individual or composite) directly	*Rec. 4: Develop governance processes that support service consumers in the identification of use cases and the tests necessary to achieve appropriate assurance about performance of the service within those use cases.*
Service integrator	Uses existing services (individual or composite) either to create composite services or to create an end user application	*Rec. 5: Develop guidelines for testing composites of various types, including composites that implement capabilities supporting pre-defined mission threads and ad hoc composites that reflect attempts to address emerging needs.* *Rec. 6: Develop guidelines for testing composites employing the range of composition mechanisms expected (e.g., WS-BPEL defined, application-embedded composites, WS-CDL).*

[5] The table is an extension of roles identified in "Testing services and service-centric systems: challenges and opportunities" [66].

Role	Action	Recommendation for SOA Testing
Infrastructure provider	Provides the necessary SOA infrastructure middleware (e.g., ESB) and infrastructural mechanisms such as service discovery to service providers, service consumers, and service integrators	*Rec. 7:* *Develop guidelines and governance processes for testing/verification of new and revised infrastructure capabilities, including notification of users of infrastructure changes and triggers for retesting.* *Rec. 8:* *Develop policies for the type and level of testing support provided by the infrastructure provider to the service provider or integrator.*
Third-party service tester or certifier	Validates and potentially certifies whether a service (individual or composite) works as expected	*Rec. 9:* *Identify the focus, expectations, and limitations of third-party testing/certification activities.*
End user	Uses applications that employ services	*Rec. 10:* *Develop policy clearly delineating the forms of testing performed according to SOA roles.*

3 Testing Challenges

Testing a service-oriented system is made challenging by a number of factors, not least of which is that most such systems are not only componentized, but also distributed. In addition to the traditional problems of distributed systems such as communication failure, the nature of service orientation, where different components may be provided by different organizations leads to additional challenges. We have already identified the various structural components involved and now discuss the challenges that each presents.

3.1 SOA Infrastructure Testing Challenges

We have identified five factors that complicate testing of SOA infrastructure: limited technical information about components, complex configuration of an infrastructure, rapid release cycles for components in an infrastructure, lack of a uniform view, and variation across an infrastructure.

- The infrastructure provider rarely has access to the design information and source code that is desired for systems with strict quality requirements (e.g., high performance, security, safety, etc.). While systems with such requirements may choose components from a collection of previously accredited systems, even systems with less-stringent quality requirements will face the challenge of testing "black box" components.
- An SOA infrastructure consists of a number of software components such as registry, repository, ESB, and databases. Typically, most of these are COTS components, rather than custom components developed to satisfy specific functional or quality requirements. Whenever the SOA infrastructure is composed of components from different vendors—and even when it is provided by a single vendor—individual components have unique installation, administration, and configuration requirements. Each component must be configured correctly for testing, and testing results can vary across multiple configurations of the same infrastructure components.
- There may also be multiple patches and service packs available for individual components which must be aligned. For example, a quick search performed for this report identified seven different software releases (versions, service packs, vulnerability patches) for one common ESB during the 13-month period from January 1, 2008 through January 31, 2009. This rapid cycle of updates necessitates an infrastructure testing process that is fast and efficient.
- Many individual components in the SOA infrastructure have their own applications for managing, data recording, and optimizing the component. A single unified view that supports the tester is often not available.
- Implementations of web services stacks (the infrastructure that supports various web services standards) can be inconsistent regarding the COTS, government off-the-shelf (GOTS), open source, and other products employed; the subsets and versions of SOA standards implemented; and other characteristics. Because COTS products are black boxes, such inconsistencies can complicate testing by introducing factors, such as different error handling methods and inconsistent use of standards that are not understood clearly by developers and testers.

3.2 Service Challenges

The following are key testing challenges from an individual (atomic) web service perspective.

- *unknown contexts and environments*: One of the biggest advantages of web services is their reusability in variety of contexts. From a service developer's perspective (that is, having responsibility for unit testing the service), it is not easy to anticipate all situations where a service may be used. This lack of knowledge about usage contexts hampers testing, although being aware of all contexts is not necessarily a solution either. To address this problem, it is necessary to determine a set of contexts (based on best estimates) and prepare tests in these contexts.
- *unanticipated demand and impact on quality of service*: Unlike traditional software components, a single instance of a web service can be used by multiple consumers. Since the specific usage of the service (e.g., load, network and infrastructure delay, data) is unknown at the time of development and deployment, it is difficult to test and verify whether a service can successfully meet the quality of service (QoS) expectations of users. Doing so requires testing for performance under different conditions, such as varying loads.
- *lack of source and binary code*: In many cases, the source code and binary code [16] of the service and its underlying components are unavailable to the service integrator and tester. In the absence of source and object code, white box testing techniques such as static analysis are impossible.[6] This circumstance is particularly problematic for organizations that maintain high information assurance standards, since it becomes difficult to verify that a service is not acting in a dangerous manner. In most cases, these loosely coupled services are black boxes. Design and code-level information is often not be available to service providers, meaning that subtle assumptions embedded in the code can only be identified through rigorous testing.
- *standards conformance*: Web services are based on standards that facilitate syntactic interoperability. These standards are defined at a high level of abstraction and are required to conform to other complementary standards [17]. This level of abstraction leads to significant variation in actual vendor implementations, creating additional work for the service tester to verify compliance with standards such as SOAP, REST [18] , XML, UDDI, WSDL, and HTTP.[7] Tools support testing of many of these well-established standards and they should be used as appropriate. However, tool support is lacking for several recommendations and standards that are newer and less widely adopted (e.g., WS-Notification [19], WS-Trust [20]).

In addition to the challenges posed by testing individual services, the following challenges accompany testing composite services and end-to-end threads:

- *service availability*: Composites use services that may be controlled by multiple service providers, and testing of a composition requires that the correct versions of all invoked services are available. Therefore, the schedule for testing the composite is strongly dependent on schedules for other services outside the control of the testing organization.

---

[6] White box or glass box testing usually is performed with the knowledge of the internals of an SOA element.

[7] Acronyms used in this report are identified in Appendix A.

- *multiple provider coordination*: Even when a service integrator has access to all participating services and components, the individual services may be black boxes to the integrator. Thus, the service integrator may need to communicate with the teams providing these services in order to learn about the implementation characteristics of the service. For example, if a participating service saves information in a database that is not visible to the integrator, the integrator will need to work with the service provider in order to get access to that information. While access to such information may not always be necessary, it will be a common enough occurrence that the testing will likely encounter need like this in any significant system.
- *common semantic data model*: There will be a large potential for misunderstandings of the data exchanged unless all services participating in a composite agree on a common semantic data model. However, achieving consensus on common semantics is a notoriously difficult job, and it's critical for the tester to determine that the applications create data consistent with the data model and that they transform it according to specified rules. Inconsistencies in how data is understood can be very subtle and show up only in a few situations; as a result, they may be difficult to identify. For example, consider that two services may use temperature but one measures in Celsius and the other in Fahrenheit.
- *lack of a common fault model:* Given that services will likely be developed by different service developers, for different purposes, it is unlikely that they will subscribe to a common fault model. Reconciling fault models is as hard as achieving a common data model and leads to difficulties when testing that service faults are handled appropriately.
- *transaction management*: Different service providers and SOA infrastructures may implement slightly different transaction management approaches (e.g., they may have different models of when a change to a data item actually occurs). These differences may only show up in unusual or race conditions where the split-second timing of multiple transactions leads to inconsistent states or service failure. Therefore, the correctness and consistency of all transactions must be tested.
- *side effects*: Because a single service can be part of multiple composites, execution of test cases for one composite may produce unwanted side effects in others [21] employing the service. Also, changes in data or state caused by testing can interfere with other service composites or produce unwanted changes in them. Of course, not all services will suffer from interference between threads; indeed, it is desirable to develop services without such interference. However, the difficulty for the tester is determining that any given service is free of side effects.
- *independent evolution of participating services*: The service integrator often has no control over individual services or their evolution, yet any change in a participating service can result in a failure of the composite. If it is unaware of service changes, the service integrator may be unable to identify the source of a failure. Even when it is aware of the changes, the service integrator still needs to perform a set of regression tests to ensure the proper functioning of the composite.
- *lack of tool support*: Current testing tools and technologies that claim support for composites often provide the ability to validate whether a business process representation (e.g., WS-BPEL) is well-formed and check for the existence of the endpoints of the individual web

services that make up the composite web service. Tool support for more comprehensive testing of composites is an active research area.

3.3 Environment for Testing

A final challenge, but one that is necessary to overcome, is the creation of a testing environment that is as similar as possible to the deployment environment (a high fidelity copy) that allows

- developers to test and integrate services and service orchestrations prior to actual deployment
- testers to design, implement, execute, and manage the testing process,[8] and to capture information about the functional and quality aspects of SOA services, infrastructures, and end-to-end threads

Establishing a testing environment is difficult because many deployment environments are widely distributed, with heterogeneous hardware and software (SOA infrastructure, services, operating systems, and databases). Because of cost and security, it may not be possible for all organizations hosting a capability in the deployed environment to mirror those capabilities in a testing environment.

An alternative involves developing a testing environment that mirrors several key aspects of the deployment environment and provides workarounds for missing aspects. In some cases, test copies of deployment infrastructure may be available; in other cases, virtual machines can be stood up to mirror parts of the infrastructure. This testing environment should be available to developers for early testing of services and service orchestrations, as well as to testing organizations performing certification activities. The testing environment should be populated with

- a suite of testing tools to manage tests, validate conformance to standards, and analyze runtime functionality and performance
- test data and associated databases sufficient to support the execution of tests
- a mechanism to determine the testing/certification status of a service or service orchestration (end-to-end thread)
- versions of deployed infrastructure and services sufficient to support the execution and testing of services/service orchestrations under development and test
- a mechanism for stubbing out or simulating services that are not yet developed

Recommendation for SOA Testing

Rec. 11: Provide developers with a fully outfitted test environment for early testing of services and service orchestrations; provide this environment to testing organizations, too, for performing certification activities.

[8] While this report is not about defining a process for testing SOA implementations, we have included some thoughts about testing process inputs and outputs in Appendix D.

4 Testing Functionality

4.1 Testing SOA Infrastructure Capabilities

An SOA infrastructure provides capabilities, often in the form of common services that can be used by a number of business processes. Services are usually composed of COTS components that are tested by their vendors, which may mask key information (such as the test cases or their context) from the infrastructure provider. Recommendations for testing SOA infrastructure capabilities are shown in Table 3.

Table 3: Recommendations for Testing Selected Infrastructural Capabilities

Infrastructural Capability	Implication for SOA Testing	Recommendation for SOA Testing
Service Registration Mechanism	The goal of testing service registration is to ensure that only validated services become part of the SOA registry. A final testing results report is sent to the service provider once the automated testing is complete. It is also possible for the service providers to submit test cases conforming to a standard when they submit a service for registration. Testing the service registration mechanism involves testing the tools and options provided by the infrastructure to register a new service. It also needs to take into account the version control of services already registered.	*Rec. 12*: *Develop strategies for testing of the service registration mechanism.*
Service and Resource Discovery Mechanism	Testing the service and resource discovery mechanism ensures that registered services and infrastructure resources can be found by authorized service consumers. Testing also needs to be performed if the service consumers can search for the services they require using various criteria and matching rules.[9]	*Rec. 13*: *Develop tests to determine whether authorized service consumers can find services based on syntactic or semantic approaches.*
Subscription and Notification Mechanisms	If the infrastructure provides subscription mechanisms, then these need to be tested. All service consumers must be informed when a new version of a service is registered with the infrastructure. Service providers and consumers also require notifications of service level violations.	*Rec. 14:* *Develop tests that reveal whether all appropriate service providers and consumers are notified when a new version of a service is registered with the infrastructure or when service levels are violated.*

[9] A discovery mechanism can be syntactic or semantic. Syntactic discovery, while easy to implement and test, has limitations because service consumers may not always be aware of the keywords to use in a search, and the use of just keywords to describe services often ignores their operational characteristics. A semantic approach promises a greater probability that services sought will be found, but requires more testing effort on the part of the infrastructure provider which must test various matching rules for service discovery.

Infrastructural Capability	Implication for SOA Testing	Recommendation for SOA Testing
Service Usage Monitoring	Service usage data is a source of information for service providers in identifying who will be affected by changes to a service. Monitoring provides data important for audits and planning (e.g., security audits, planning for increases in load).	*Rec. 15:* *Develop a testing strategy for service monitoring capabilities.*
Service Virtualization	Service virtualization creates and manages virtual endpoints for web services, allowing the service to be dynamically associated with physical endpoints in scenarios such as load-balancing.	*Rec. 16:* *Develop a testing strategy for service virtualization features.*

4.2 Testing Web Service Functionality

Services provide the main functionality or capability in an SOA environment. An initial step in testing is to determine whether each service meets its functional requirements. While testing the functionality of SOA services uses standard component testing methods, it poses a greater challenge because of the lack of control over many aspects of the test situation, including the services, SOA infrastructure, and other components with which the service being tested must interact. This section highlights approaches to address the challenges of testing service functionality.[10]

The use of stubs to "mock" the behavior of unavailable components is a common software testing strategy. This method has applied to the testing of services that need to interact with other services where not all of the dependent elements are in place. Wolff, Godage, and Lublinsky [22,23,24] show how mocking can be used to test services. Both open source (e.g., Apache Synapse[11]) and commercial implementations (e.g., SOAPSimulator, soapUI) can help service developers to create mock services for unit testing quickly because not all functionality needs to be tested.

However, this approach has certain limitations including the following:

- The quality of the testing result depends on the fidelity of the mock service to the actual service. Services that behave in a complex manner are difficult to mock well, because understanding the complex behaviors and implementing these behaviors with testing tools is itself a complex task. This leads to practical limitations of the types of services that can be mocked.
- The behaviors and interfaces of services to be mocked are often not finalized since development is not complete: this is why the services must be mocked. This complication inevitably leads to uncertainty in the reliability of testing results.
- QoS cannot be adequately tested if services or other environment components are mocked.

[10] Appendix C contains lists of attributes for testing functionality and other aspects of web services.

[11] For more on Synapse, go to http://synapse.apache.org/.

Recommendations for SOA Testing

Rec. 17: Provide support for service mocking, including mocking of services on remote platforms and infrastructures.

Rec. 18: Integrate service mocking into the development and testing strategies.

4.3 Fault Injection

Fault injection, when used as a standard software testing approach, makes deliberate errors in the implementation to see if the system can detect the fault and recover from it. When applied to service testing, fault injection focuses on determining that services don't produce unintended behaviors or go into unreachable states when unexpected or faulty inputs are provided. This technique can be applied to both individual web services [25] and composites [26].

Fuzzing is a testing technique supported by many tools that generates invalid data inputs to web services. The tools capture the results of the "fuzz" and support the analysis of results to determine whether it compromised the web service.

Examples of types of faults that can be injected are a malformed service interface [27] (e.g., bad WSDL definition), perturbation in the message [25] (e.g., null values in a SOAP message), and timing delays in a composite (e.g., a service may timeout if it doesn't get a response). Fault injection is also used to test the security, reliability, and performance of a service as discussed in Sections 6 and 7.

Recommendation for SOA Testing

Rec. 19: Develop guidelines for the type of faults that services and service compositions must be tested for.

A standard testing practice is testing the software to determine whether it handles realistic data appropriately; such testing applies equally to services. In order to access appropriate test data, the SLA should document the responsibilities for producing test data sets and appropriate test interfaces. As always, care must be taken to obfuscate production data since web services undergoing testing may not assure the same degree of security as production services, nor are the persons performing the testing guaranteed to be authorized to access production data.

Recommendation for SOA Testing

Rec. 20: Formal agreements should be in place with data providers to identify test data sets and test interfaces.

4.4 Regression Testing

Because of the rapid evolution of an SOA environment, testing cannot be a one-time event. Services, service composites, end-to-end threads—and applications using them—continue to evolve. This section focuses on the specific aspects of regression testing that need to be considered in an

SOA environment. Regression testing involves testing and identifying bug fixes to ensure that defects have not been introduced as a result of evolution. Regression testing also attempts to develop some level of assurance that things that were not touched did not break.

Regression testing in SOA environments is challenging because every service composition, business process, and other system using a particular web service is affected by changes made to constituent service [21], unlike a component-based context where the new release of a component affects only the new versions of that component.

A service baseline aids regression testing efforts by making it easier to determine whether new functionality has introduced defects and to more quickly mitigate the effects of those defects. Testing tools that provide regression testing capabilities for web services offer the ability to take a service baseline snapshot, so that subsequent changes can be automatically detected and reported.

Changes in that baseline can serve as triggers that indicate the need to perform regression testing of end-to end mission threads. Several of these triggers are

- upgrades/version changes to requestor/responder application, infrastructure, services invoked, back-end systems
- upgrades to Service Contracts (WSDL) or SLAs (e.g., changes to expectations for the services within the mission thread)
- changes or revisions to life-cycle management of components
- changes in deployment configurations
- changes to back-end systems or data sources
- changes in a service's functional behavior or QoS
- updates to the rules used by testing tools
- deprecation or retirement of a service

An SOA test sandbox is an infrastructure that accurately mirrors the deployment environment. A test sandbox not only makes regression testing easier, but it also allows the use of automated testing tools that can perform regression testing as soon as new or updated services are deployed after a successful build process. Service developers can review the results of the automated testing tools and initiate mitigation where appropriate.

The actual determination of whether to perform regression testing will be based on many factors, including the nature of changes and the profile of the mission thread. Clearly, an organization will be more risk-averse with, and therefore more likely to perform regression testing on threads that can affect human life or cause widespread environmental or economic damage.

Recommendations for SOA Testing

Rec. 21: Establish baselines for services, compositions, and end-to-end threads.

Rec. 22: Establish regression testing triggers for each.

Rec. 23: Establish a regression testing sandbox fitted with appropriate testing and analysis tools to automate regression testing to the degree possible.

4.5 Testing Business Processes That Span Traditional Boundaries

End-to-end testing in an SOA environment refers to testing of the entire pathway involved in executing a specific mission or business process. For one example, in a military application, Time Sensitive Targeting (TST) is often considered to have six phases: find, fix, track, target, engage, and assess. These activities may span a number of application and organization boundaries. They cover the point from which a potential target is identified to the point when it is no longer considered a threat. End-to-end testing of this capability in an SOA environment would involve executing a number of services, applications, and input from humans. The invoked SOA services may be provided by different U.S. military services and other agencies, and possibly by coalition forces.

End-to-end testing needs to consider:

- *Decentralized ownership and lack of centralized control*: Distributed ownership in an SOA environment often involves setting up appropriate data and process context in back-end systems that are outside the control of the end-to-end tester. Also, services may be created by different service providers, hosted on different infrastructures, and used by a variety of consumers. These complications introduce the need for creating an environment (through negotiation) for testing, testing across common infrastructures, testing across disparate infrastructures, and coordinating testing with multiple service providers and owners of back-end systems.
- *Long-running business activities*: SOA implementations often support the business processes and workflow that constitute long-running business activities. For example, a loan approval business process could be executed in phases over weeks.
- *Loosely Coupled Elements*: Web services deployed in an SOA environment are loosely coupled; they make minimal assumptions about the sending and receiving parties, so that changes in one entity will have minimal impact on the other interfacing entities. The advantage of such loose coupling is that testing becomes an issue of testing the composition through simple interactions rather than complex ones.
- *Complexity*: An end-to-end implementation is usually an execution pathway though multiple people, services, composites, and infrastructure capabilities. Information—about specific services invoked, parameter data passed, and attachments included—is embedded in protocols that are intended to support machine-to-machine interaction.
- *Regression testing*: Changes at any service, node, or component along the pathway exercised in support of an end-to-end thread may indicate a need for regression testing of the thread. Maintaining awareness of these changes requires agreements regarding what types of changes require notification, when such changes are allowed to occur, and how affected parties are notified. An approach to bounding the need for regression testing is that of assurance

cases (see Section 9.6) which provide a documented chain of reasoning based on evidence to support a claim. In this case assurance cases could be used to help identify the conditions under which further regression testing would be required.

Recommendations for SOA Testing

Rec. 24: *Use approaches such as assurance cases to identify when additional regression testing is needed.*

Rec. 25: *Develop a strategy for testing at each of the layers of the SOA interoperation stack.*

Rec. 26: *Instrument test instances to help identify the sources of failures.*

Rec. 27: *Perform regression testing to identify the impacts of changes on services, nodes and components along the pathway that is exercised.*

4.5.1 Process for End-to-End Testing

The basic process for testing end-to-end threads in an SOA environment does not differ from the high-level steps involved in other testing processes, with one exception: not only must tests be devised to establish the correctness of predefined mission threads, but additional tests must be devised to address the potential for dynamically composed mission threads (see Figure 1). Some details of the steps are unique to SOA environments, as indicated in subsequent sections.

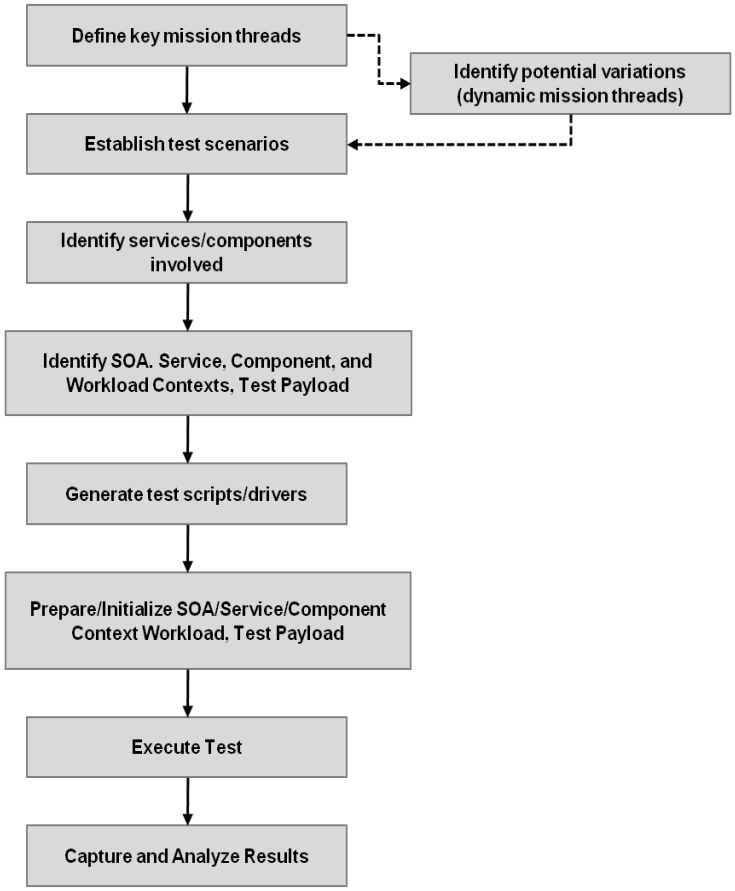


Figure 1: Mission Thread Testing Process

4.5.2 Defining End-to-End Threads and Establishing Test Scenarios

With SOA—as with any other technology that has many possible threads of control—critical threads must be identified and specifically tested for both functional and quality characteristics. Several general approaches are used to identify end-to-end threads and establish test scenarios for SOA environments. Manual approaches analyze mission functions, decompose these functions into key decision steps and associated conditions, and define and build test cases accordingly. The following process steps, derived from Sikri [28] are representative of manual processes:

1. Build a tree of high-level functions including inputs, outputs, and execution paths.
2. Identify conditions (e.g., timing, environment, and data) for each function in the tree.
3. Establish test scenarios based on functions and conditions.
4. Build tests cases addressing functions, conditions, and scenarios.

Other techniques are often needed to understand quality of service attributes and relate them to various scenarios. A technique such as the Carnegie Mellon® Software Engineering Institute (SEI) Mission Thread Workshop[12] elicits quality attribute needs associated with the end-to-end threads

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

12 For more information about the Mission Thread Workshop, go to http://www.sei.cmu.edu/architecture/tools/evaluatingsos/mission.cfm.

of a systems-of-systems environment [29]. However, for certain types of services (e.g., public web services on the Internet) where the actual operating context (e.g., expected load and usage, level of security required, etc.) is unknown until the service is actually being used, it may be necessary to adopt an incremental approach in which functionality is fielded and the system is then adjusted to meet QoS goals.

Automated approaches have recently been developed that model mission threads in formal languages or that capture mission threads and establish test cases from existing modeling language (e.g., Unified Modeling Language [UML]) artifacts. The resulting structures are suitable for many types of analysis (e.g., dependency analysis, consistency checking, and test case generation) [30,31,32]. While these approaches have significant potential for supporting rapid analysis of dynamic mission threads, the modeling language techniques are still in the maturing stage, so using them needs to be done in small settings and with careful attention to results.

Recommendations for SOA Testing

Rec. 28: *Develop a strategy for identifying the important threads in an end-to-end threads and decorating those threads with both functional and QoS expectations.*

Rec. 29: *Develop test cases for critical threads that have been identified.*

Rec. 30 *For certain types of services (e.g., public web services on the internet) where the actual operating context (e.g., expected load and usage, level of security required, etc.) is unknown until the service is actually being used, it may be necessary to adopt an incremental approach in which functionality is fielded and the system is then adjusted to meet QoS goals.*

4.5.3 Identifying Dynamic Mission Threads

We believe that at this stage of SOA maturity, the use of dynamically composed mission threads in SOA environments has been oversold. Dynamic composition requires alignment at several layers of the interoperability stack (see Figure 2 for details) for the services and applications invoked. Requisite standards supporting such alignment are not yet available. As a result, dynamic composition in current SOA environments will rely on local conventions and standards that are likely to be inconsistently defined and hard to enforce across organizations.

Dynamic composition is more likely among services and applications that are recognized as related and for which local conventions and standards have been established and tested. We recommend that for SOA environments where dynamic composition is anticipated, every effort should be made to identify and test likely compositions. Such testing is necessary to identify not only successful operation of the dynamic thread, but also the effects of dynamic threads on network, SOA infrastructure, individual services, and other critical threads.

Recommendations for SOA Testing

Rec. 31: *Develop a strategy for identifying and testing likely dynamically composed threads.*

Rec. 32: *Implement "non-interference" strategies such that other threads cannot compromise functioning or QoS in the rest of the end-to-end thread.*

4.5.4 Identifying and Initializing the Context and Test Payloads

Establishing the testing context for end-to-end thread tests is a well-known problem in traditional testing and continues to be a problem for SOA testing. It involves realistic simulation of the environment in terms of capabilities, data, and loading. This section focuses on the specific aspects of context to consider in end-to-end testing in SOA environments.

SOA environments often depend on a specific data value or other response from a black box service. Well-understood principles that need to be continued are to (1) start with agreements with service providers concerning the type of test data available, as well as rules governing testing of the services, (2) use tools to simulate the production environment, (3) use data sets with high fidelity to enable testing such QoS attributes as performance, and (4) establish realistic loads on infrastructure components (network and SOA) as well as on the invoked services and related applications.

Recommendation for SOA Testing

Rec. 33: Develop a strategy for cross-site and cross-organizational collaboration to identify test context and data loads.

4.5.5 Capturing and Analyzing Results

End-to-end processing requires that the collaborating services, infrastructures, and applications not only execute an action as directed but also that the action performs the correct activity. Capturing data regarding the performance of services invoked in an end-to-end test may require instrumentation of the services or infrastructures on which they execute. The format of this metric data differs according to the characteristics of the infrastructure and tools employed. These challenges can be addressed by making specific decisions on the data that is to be captured, by providing a logging service and by using a specially instrumented infrastructure.

An automated test tool offers value by collating and consolidating the test results to provide a single point of review, making it easier to identify, categorize, and publish defects.[13] Testing tools that are a part of a comprehensive design and development suite offer the added benefit of linking test plans to requirements, ensuring that the notion of requirements traceability is satisfied.

Recommendation for SOA Testing

Rec. 34: Develop a strategy for cross-site and cross-organizational verification of changes in back-end systems and capturing and analysis of test results.

[13] In addition, a recent report from the Aberdeen Group [62] suggests that automated tool support for testing in SOA environments is a critical differentiator between the most sophisticated employers of SOA ("best in class") and those that are less sophisticated. Only the related practice of *through-life quality management* was cited by more best–in-class organizations as being critical to successful SOA.

5 Testing For Interoperability

A fundamental assumption of SOA adoption is that the resulting collection of services and infrastructures will provide the basis for the creation of new end-user applications or end-to-end threads through reuse of existing services. A consequence of this assumption is the reliance on the interoperation of services that weren't specifically designed to interoperate. Interoperation of the individual components in any distributed environment (in this case, an SOA environment) relies on agreements at several levels between service requestors and service providers, which makes the execution of mission threads difficult. A further complication is that interoperation is the interaction *between* two entities whereas testing traditionally focuses on entities and not their interactions.

5.1 Defining Interoperability Levels

To achieve end-to-end testing, the services have to be interoperable. Figure 2 identifies the types of agreements that must exist between service consumers and service providers to ensure successful interoperation.

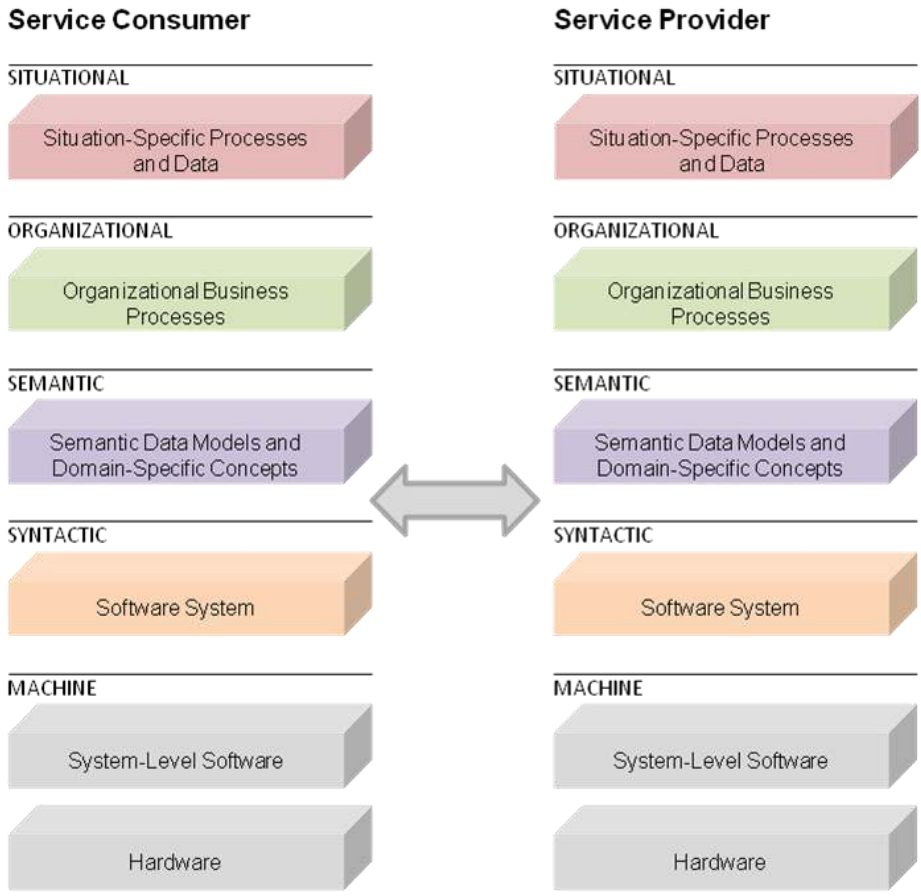


Figure 2: SOA Interoperation Stack Showing Levels of Interoperability

Agreements need to be made between service providers and service consumers at all levels. The agreements include the following (reading Figure 2 from the bottom up):

- *Agreements at the Machine level*—Protocols for network-to-network and machine-to-machine interaction, such that -two heterogeneous systems can exchange data irrespective of their hardware, system-level software, and network characteristics. For example, an application running on an Army tactical network can gain access to and interact with a service system running on an Army enterprise-wide network. This level of interoperability calls for physical, data link, networking, transport, and session layers of the Open Systems Interconnection Basic Reference Model (OSIRM). [14]
- *Agreements at the Syntactic level*—Alignment of the structure and syntax of data and control structures and interactions of applications and services. Within the web services domain, the major standards include WSDL, SOAP, and, more recently, REST.
- *Agreements at the Semantic level*—Semantic meaning at the application level on data and control interactions among services. These agreements may reflect shared ontologies across the interacting components (services, back end systems, etc.) or mechanisms for mapping/translating between different semantic models. Technologies used by web services to establish common semantics for data include general purpose languages like the RDF and specialized vocabularies like. OWL, OWL-S,[15]and WSMO[13] have been used to define computer-interpretable definitions of service semantics. The same concept can apply to control models. For example, for services implementing transactions to successfully interact, there must be shared semantics for transactions (such as WS-AtomicTransaction or WS-Coordination) or a mapping across individual transaction models. Unfortunately, technologies that provide semantic consistency are not as well developed, stable, or accepted as agreements at the lower levels of the SOA interoperability stack.
- *Agreements at the Organizational level*—Characteristics of predefined mission threads or business processes, including the associated context and workflows of thread participants, as well as the appropriate qualities of service required. Some of the necessary agreements at this level are defined in SLAs. WS-BPEL also provides a language to define an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the WS-BPEL designer [33].
- *Agreements at the Situational level*—Characteristics of specific situation in which mission threads or business processes are being applied. These characteristics are always unique to a specific context. For example, consider a soldier calling in air support. While there are existing processes that are executed when calling in air support, the specifics of the urgency with which air support is dispatched and how it is delivered will depend on the proximity of the enemy, air assets available, and many other factors. There are no public standards for this specific contextual information, but there may be less formal semantics and heuristics that are applied.

---

[14] Acronyms used in this report are identified in Appendix A.

[15] Formerly DAML-S; OWL-S is the Ontology Web Language for Services.

Only the lowest two layers of the stack (machine and syntactic alignment) are addressed by commonly used web service standards. However, the standards that exist allow for testing services in isolation for their conformance to those standards.

5.2 Interoperability in Infrastructure

The SOA infrastructure provides components for the lower two layers of the interoperability stack and, as such, is most amenable to testing for conformance to well-defined standards such as those provided by W3C. Such conformance testing is discussed later (see Section 8); however, there is other infrastructure testing to be performed. Specifically, it is important to test that the components within any one infrastructure interoperate appropriately and, where needed, that there is appropriate interoperability between infrastructures.

5.2.1 Testing Within the Infrastructure

Typically the SOA infrastructure will be composed of one or more COTS products that have been brought together to form a coherent infrastructure satisfying the organization's needs. The products should be selected through an evaluation process that enables the developers to select the products most suited to satisfy the requirements. The test community can participate in the evaluation process using their expertise in testing as the mechanism for evaluating the products. Such evaluation should consider not only the features of the individual products but also how well the products fit with the rest of the candidate or already-selected products. Once the infrastructure has been assembled, testing can be used to develop confidence that the infrastructure meets the needs. We have already discussed testing the functionality of the infrastructure (See Section 3.1). However, additional testing could include testing for other quality attributes as well as simple assurance that the infrastructure provides the basic connectivity between service consumers and providers.

5.2.2 Testing Between Infrastructures

Increasingly, SOA infrastructures are joined with other SOA infrastructures; it becomes essential to identify these configurations and to test to ensure that they federate as required. Some federation configuration types are

- *homogenous infrastructure*: This is essentially a horizontal scaling of the SOA infrastructure where the federated infrastructures are different instances from the same base. Testing this federation primarily involves checking connectors between the instances. For example, if the same ESB product is used by all infrastructure nodes, the main testing task is to ensure that the connection between two instances of an ESB by the same vendor functions as desired.
- *heterogeneous infrastructure*: This configuration contains nodes that are different—sometimes the difference may only be in terms of versions; it is more common than the homogeneous type because nodes are often planned, implemented, and upgraded separately.
- *hybrid infrastructure*: In this configuration, some nodes are identical and others are different; testing will require tactics from both homogeneous and heterogeneous infrastructure testing.

Regardless of the nature of the federation, it is important to perform the testing that would be applied to a single infrastructure, but across the federation in order to provide confidence that the federation behaves as a single "virtual" infrastructure. Capabilities of particular importance are

federated identity management and discovery. Without appropriate interoperation between the infrastructures in at least these areas, the federation will not achieve the organization's aims.

Recommendation for SOA Testing

Rec. 35: Develop a strategy to identify and test the capabilities of SOA infrastructures that are needed to federate the infrastructure.

5.3 Interoperability in Services

Services are found at both layers two and three of the interoperability stack (Figure 2), with infrastructure services tending towards being in layer two and services providing business functionality tending towards layer three. Regardless of the source of the services (infrastructure or business), testing them for interoperability can be handled in the same way, through means of syntactic, semantic, and end-to-end interoperability testing. It should be noted that interoperability is a property that exists between two services and is not a property inherent in the services themselves; thus testing services for interoperability becomes largely a matter of testing the service interfaces' adherence to standards.

When web services are being used, testing for syntactic interoperability can be performed by determining that the services conform to the appropriate standards (see Chapter 8). Even when web services are not being used, the SOA infrastructure will have a defined collection of protocols that it supports and syntactic testing is an issue of determining that the service adheres to those protocols.

Layer three describes semantic interoperability between services, which poses a problem for testing the meaning of service interoperation. In order for services to be semantically interoperable, they must perform according to expectations, thus a basis for interoperability is that each service meets its specification, such testing has already been discussed in Section 4.2.

A requirement, though, for two services to be able to interoperate semantically is that they agree on the data to be communicated. To the extent that the data models are well described, either in common ontologies or in ontologies that can be mapped to each other, the test community can test the potential for interoperation by ensuring that the service interface adheres to the defined model(s). Clearly this isn't sufficient for full interoperability testing; however, even such minimal checking will be of benefit to the service developers.

Recommendations for SOA Testing

Rec. 36: Develop a list of interoperability standards that services must comply with, and consider offering an interoperability compliance check as a service to developers.

Rec. 37: Develop a list of data models that services must use and consider offering a compliance check as a service to developers.

5.4 End-to-End Interoperability

End-to-end interoperability may be found at the top two levels of the interoperability stack (Figure 2). At level four, two or more organizations align their business processes with each other and an end-to-end thread may cross the organizational boundaries, invoking services owned and

operated by each other. Level five of the stack allows for the development of end-to-end threads in a dynamic fashion in order to meet a specific demand. No matter how the thread arises, the issues for testing are the same: to ensure that a sequence of discrete steps (the services) can be invoked even though they are geographically separate and owned by different organization and that the invocation leads to desirable behavior.

Given that the number of services is potentially large, the number of possible interactions grows exponentially, and not all interactions could have meaning, it is clearly not possible to test all interactions to ensure that the services are semantically interoperable. Testing interactions should be largely performed in some context, specifically that of known end–to-end threads. Composite services, such as those defined in, say, Business Process Execution Language (BPEL), are a special case of end-to-end threads. The interfaces to composite services can be tested in the same way that services are tested, and the interactions between the services within the composite can be tested in the same way that end-to-end threads are tested.

As a result, it is difficult to ensure consistency at the semantic and organization levels.[16] The most common strategy for achieving consistency at these levels is to develop predefined mission threads (i.e., static threads) and test to ensure that consistency is achieved in use of vocabulary and organizational interactions. Consistency at the situational level will often involve human-to-human communication of the nuances of specific contexts and therefore it is difficult (or more likely impossible) to make entirely consistent.

To address these challenges, a set of guidelines can be followed that includes

- thoroughly test the services using the techniques defined in Section 4.2
- test adherence to data standards
- functionally test the defined threads
- test the threads for performance, security, and other quality attributes

Failure at any of level in the stack can lead to a failure of mission thread. However, up to this point (i.e., for predefined mission threads) an SOA environment does not necessarily differ significantly from other distributed environments. It is possible to tailor individual capabilities and services, back-end systems, and network and SOA infrastructure to support the needs of specific mission threads. For example, some organizations are tailoring the SOA deployment architecture by adding additional hardware (e.g., hardware accelerators for XML parsers) or standing up additional SOA infrastructure. Taking these kinds of steps enables SOA end-to-end testing to be conducted in ways similar to testing in any distributed environment.

A big selling point of SOA has been the ability to dynamically compose services in response to specific, emerging situations. Scenarios are commonly presented where personnel check a registry to determine what services are available and quickly craft and deploy a unique solution perfectly matched to an emerging problem. However, dynamic composition (and late binding in general) provides an exceptional challenge for verifying interoperability. The activities to verify syntactic, semantic, and organizational interoperability must be performed "on the fly" against the specific scenario presented by the dynamic composition. At best, only testing will only show that each

[16] While semantic alignment is being addressed by additional web service standards (e.g., OWL-S, WS-Coordination), these standards are not widely used.

service conforms to the standards in the system including, ideally, to a common vocabulary (or vocabularies that can be mapped to each other).

Standardized mechanisms to support the dynamic composition of the semantic, organizational, and situational aspects of interactions are either immature or simply not available. Therefore, any dynamically created end-to-end threads must be analyzed to ensure that they do not become error prone and subject to inconsistencies. Those threads will not have been previously considered in the testing process, and there is little assurance that they "interoperate" beyond the very basic syntactic level provided by a WSDL specification. It is also useful to capture dynamically created scenarios for future testing and/or incorporation into future releases.

We recommend that at this point of SOA maturity, compositions should be predefined and tested for interoperability as well as other quality attributes. Mechanisms should be put in place to assure, to the extent possible, that these predetermined compositions are not compromised by dynamic, on-the-fly compositions of services. Dynamic compositions, if allowed, will likely be "use at your own risk" for the near term.

Recommendations for SOA Testing

Rec. 38: *Develop a strategy to supports scenario-driven testing of interoperability.*

Rec. 39: *Develop policies for differential treatment of predefined and dynamic compositions of services.*

Rec. 40: *Develop policies for capturing dynamically created threads for future analysis and inclusion into the collection of predefined threads.*

6 Testing for Security

SOA implementations are susceptible to many of the same security threats that afflict other networked systems. Common vulnerabilities have been found in SOA implementations (both infrastructure and SOA services), including

- denial-of-service attacks on SOAP message handlers and XML parsers
- attacks that bypass authentication and authorization mechanisms or spoof identities
- attacks that capture usernames and passwords
- attacks that execute malicious scripts (often unknown to a benign "pawn")

For more information about these and similar attacks, search the Department of Homeland Security's National Vulnerability Database [34] for *SOA, XML, SOAP,* and other common SOA terms.

6.1 Thread Modeling and Attack Surface

Maintaining a secure SOA infrastructure and services involves far more than testing and includes appropriate development and deployment practices, security monitoring during operations, and maintenance of the security profile during sustainment. However, the focus here on testing for security must start with models of the threats facing SOA implementations and potential targets of those threats—that is, on threat modeling and attack surface.

6.1.1 Threat Modeling

Threat Modeling is a technique that is often used to generate security requirements and assess possible security countermeasures, but it also provides a starting point for security testing. The STRIDE Threat Model [35] provides six high-level threat categories to identify potential sources of security failure in systems. The six threat categories in STRIDE, along with an example of an SOA threat, include:

- *Spoofing*: a malicious party taking on the identity of a legitimate web service requester, such as by submitting different faked credentials.
- *Tampering*: a malicious party altering the content of data and/or messages, for example by uploading incorrect data. Tampering attacks normally happen in the context of other forms of attacks such as spoofing that provide inappropriate access.
- *Repudiation*: a party denying the initiation or authoring of an event or transaction. Repudiation can occur when SOA transactions are not correctly logged.
- *Information Disclosure*: actively gaining access to supposedly secure information, for example through lack of (or insufficient) encryption of SOAP messages.
- *Denial of Service*: limiting or eliminating access to a system or resource. Common denial of service vectors in SOA include SOAP and XML parsers.
- *Elevation of Privilege*: taking on higher privileges than appropriate, such as a normal user taking on super user or administrative privileges. Again, problems with SOAP message handling have led to such attacks.

The purpose of a Threat Model is to provide decision support for selecting and locating countermeasures in the system. Categories of countermeasures to specific sorts of threats are identified in the first two columns of Table 4.

Table 4: Types of Threats and Typical Countermeasures

Threat	Countermeasure	Standard
Spoofing	Authentication	WS-Security
Tampering	Signature	WS-Security + XML Signature
Repudiation	Audit Logging	None
Information Disclosure	Encryption, Input Validation	Encryption: WS-Security + XML Encryption Input Validation: None
Denial of Service	Availability	None
Elevation of Privilege	Authorization	None

The third column of Table 4 identifies common standards that can apply the countermeasures. Where no widely adopted and implemented industry standard is available, the word *None* is used in the table. This does not imply that no standards exist—just that the standard is not widely implemented and adopted, as in the case of the XACML[36] standard for authorization. Many countermeasures can be implemented in various ways with differing degrees of security provided (i.e., there is significant flexibility in the standard).

6.1.2 Attack Surface

The application's attack surface comprises the ways an attacker can find a vector to attack the application [37,38,39]. Attack Surface is composed of three main elements:

- Communication channel—protocols (e.g., HTTP)
- Methods—get" and "set" methods or URIs (e.g., for SOAP, WSDL, HTTP, and for individual web service operations)
- Data— message payload (e.g., SOAP or XML)

Analysis of the attack surface can be used to locate where threats and countermeasures are applicable and should be considered in the software architecture. This report uses the concept of attack surface to assist in enumerating web services threats and countermeasures.

6.1.3 Combining Threat Modeling with Attack Surface

Since various forms of threats are possible at the many different points where an SOA implementation can be attacked, it is useful to combine the concepts of threat modeling and attack surface. The result is a matrix that identifies the various points where countermeasures may be required. Different countermeasures can be employed for SOA request and response message exchanges, and the matrix can be improved by adding this refinement. The resulting matrix is shown in Figure 3.

	Channel		Method		Data	
Threats	**Request**	**Response**	**Request**	**Response**	**Request**	**Response**
Spoofing						
Tampering						
Repudiation						
Information Disclosure						
Denial of Service						
Elevation of Privilege						

Figure 3: Threats and Attack Surfaces

6.1.4 Use of Threat Modeling with Attack Surface in SOA Testing

Figure 3 identifies many points of vulnerability that must be analyzed during SOA testing. One can actually imagine two such matrices—one for the SOA infrastructure and another for individual web services. This is a useful approach since both infrastructure and services provide their own methods and of course employ data. For example, one can imagine an attack aimed at an SOA registry (typically a capability of the infrastructure) to steal valuable data about the available services, as well as a subsequent attack on an individual service that was located via the registry.

This attack also points out another consideration: the strong relationship between the mechanisms of the infrastructure and the services. In many cases, the mechanisms of the infrastructure must be tested in concert with those of services. The mechanisms must incorporate a consistent security model (e.g., standards, standards profile, and security handlers).

6.2 Testing SOA Infrastructure Security

SOA infrastructure is composed of a somewhat variable set of capabilities, including messaging, data management, security, service registry/repository, policy management, governance, service orchestration, monitoring, and others.

Many security violations in SOA occur inside the SOA infrastructure, where an attacker slips through the network and SOA security mechanism on the perimeter and is able to gain unfettered access to the SOA infrastructure and web services [40]. While a robust testing process will help to ensure that the security risk of intrusion and compromise is minimized, it cannot replace good network and SOA security discipline. Several of the specific aspects of SOA infrastructure that should be tested against attack include

- vulnerability testing of SOA infrastructure services against common attacks
- correct creation, enablement, and operation of security policies for establishing, using, maintaining, and deleting accounts
- correct security settings for access to infrastructure data, including account information, information about services in registries and repositories, and information about policies
- maintenance of secure posture when multiple SOA infrastructures are connected (federation)
- maintenance of secure posture when services are combined into composite services (typically, orchestrations)
- correct infrastructure configuration during installation
- monitoring of appropriate and potentially malicious access
- auditing/logging capabilities€

There are many sources of information about SOA security and security controls. The National Institute of Standards and Technology (NIST) provides very useful advice but does not distinguish between testing of infrastructure and individual services [41]. Also, general information about security testing can be found in the NIST *Technical Guide to Information Security Testing and Assessment* [42]. Despite their lack of specific focus on testing SOA infrastructure, these documents provide an excellent starting point.

Recommendation for SOA Testing

Rec. 41: Develop guidelines for testing for security of SOA infrastructure, services it provides or hosts, and the environment in which it exists based on the level of security required.

6.3 Testing Web Service Security

A key challenge for the application developer is to provide end-to-end security over a possibly untrustworthy network by threading together loosely coupled, widely distributed, and potentially black box services [43]. As a result, an application developer must establish trust across a large number of distributed nodes that can be web servers, application servers, databases, ESBs, service registries, service repositories, and individual services. Many of these nodes are hidden or unknown to the application developer because service composition is recursive—that is, a service invoked by the application may invoke other services with their own set of distributed nodes, any of which could be untrustworthy.

There are several ways to reduce the attack surface of an SOA environment. An obvious way is to validate that a web service being used by an application is not introducing new resource vulnerabilities by scanning the source code for bad practices and malicious intent. There are many tools available that support source code scanning, and these tools should be incorporated into all security governance processes. However, this approach assumes access to source code for web service implementations.

When source code is available, testers should take advantage of both automated and manual techniques to verify the security footprint of web services. Testing should include verification of

- proper use of security mechanisms and standards to manage authentication, authorization, encryption, and other security aspects
- input parameters and attachments to protect against malicious content
- proper handling of exception conditions
- proper response to known attack vectors

Black box services present a greater problem for a number of reasons. Static analysis techniques (e.g., to detect potential buffer overflows) cannot be used, and isolating the problems by sandboxing is not always possible. Some obvious types of vulnerabilities may be identified by scanning service specifications for problems and testing services for various input and load conditions. Tools that support interface fuzzing are particularly necessary for testing black box services.

Clearly, the attack surface of a composite service or an application using services can be limited by reducing the number of black-boxed web services that are used. However, this strategy counters the goal of reusing as much service functionality as possible. In addition, the attack surface of

the composite or application is itself dynamic. Existing service implementations can change. As a consequence, late-binding strategies essentially remove the opportunity to completely identify the attack surface of a web service or an application composed of web services. In these cases, application developers are left only with the option of trusting service providers based on their SLAs and monitoring the service for malicious behavior.

Recommendations for SOA Testing

Rec. 42: Consider the entire attack surface in the security testing of all SOA components.

Rec. 43: Develop strategies for testing of services where source code is available and for services for which source code is unavailable.

Rec. 44: Develop an integrated security strategy that considers the network, the SOA infrastructure, web services.

6.4 Web Service Verification

Many organizations employ an additional step involving verification/certification of critical properties prior to deploying the service. The primary goals of web service verification are to analyze the service for security vulnerabilities, spot potential malicious code and un-patched vulnerabilities, and ensure compliance with security and interoperability standards. Typically, a certification authority[17] performs appropriate checks in these areas and issues a certificate for services that pass the criteria.

As an alternative to certification through additional testing and onerous checklists, this section outlines a web service verification process that develops the necessary confidence in a web service. The SEI developed this process based on ideas for the U.S. Army. The process (see Figure 4) considers the web service in terms of

- its conformance to interface standards
- the implementation of the service
- monitoring it at runtime

Interface standard checking ensures that the service meets the standards specified by the infrastructure; we say more about this form of checking in Section 8. Two different strategies are used for checking the implementation: (1) static code analysis, assisted by tools, where possible, and (2) various runtime tests such as attempts to break through the service's security. The third consideration, runtime monitoring, is a key aspect of the process and is used to provide assurance that the service continues to behave in accordance with the security policy.

This process works in conjunction with testing to maintain the confidentiality, integrity, and availability of the SOA infrastructure, other web services, back-end systems, end-to-end threads, and critical data. In this process, a web service is scrutinized from a security and information assurance perspective with respect to the following objectives:

[17] For the U.S. Army CIO/G6, the goal of this process is the verification of compliance with specific sets of standards and good practices defined by certification criteria.

- Is the web service safe? (i.e., is it safe to deploy it on the SOA infrastructure such that it will not compromise the integrity of the SOA infrastructure and other components within the architecture?)
- Is the web service compliant with the SOA infrastructure's service standards? (i.e., does it comply with the technology platform and languages, SOA standards, security and interoperability standards?)
- Does the web service have a well-defined and well-formed service contract (i.e., WSDL)?
- Does the web service include source code for review?
- Does the web service comply with the service management and governance policies mandated by the SOA infrastructure?
- Has the service provider provided the necessary service metadata and artifacts?

By forcing service providers to take their services through the verification process, SOA infrastructure managers obtain a level of assurance on the security of the deployed services. The verification process should ideally be implemented between the development and deployment phases of the service life cycle, as part of the overall governance process.

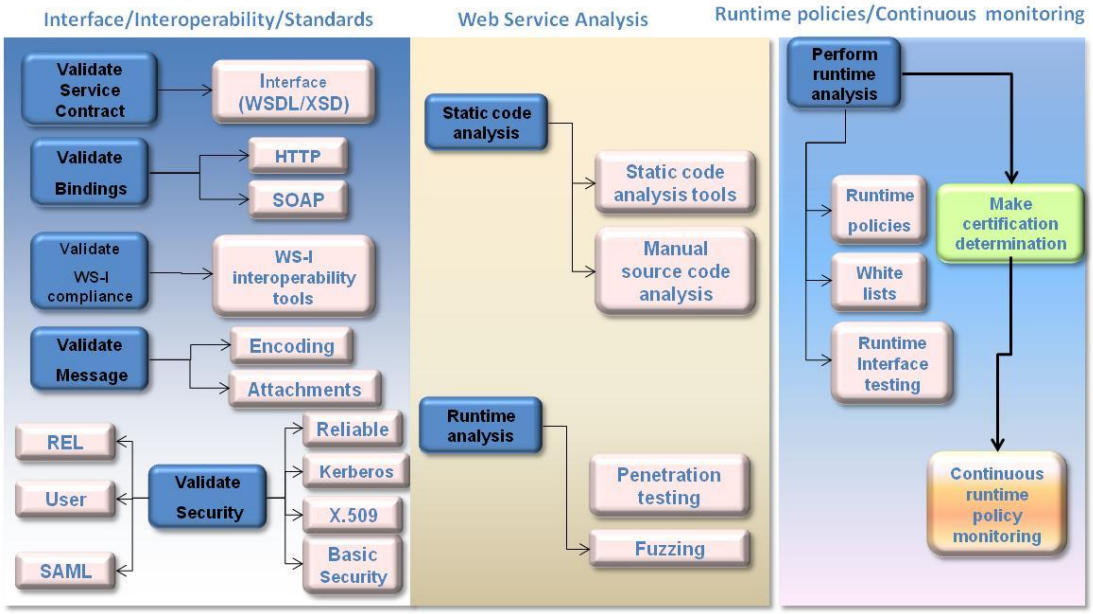


Figure 4: Overview of the Web Service Verification Process

Recommendation for SOA Testing

Rec. 45: Web service verification should be used with testing to gain confidence in the confidentiality, integrity, and availability of the SOA infrastructure, other web services, back-end systems, end-to-end mission threads, and critical data.

7 Testing for Other Quality Attributes

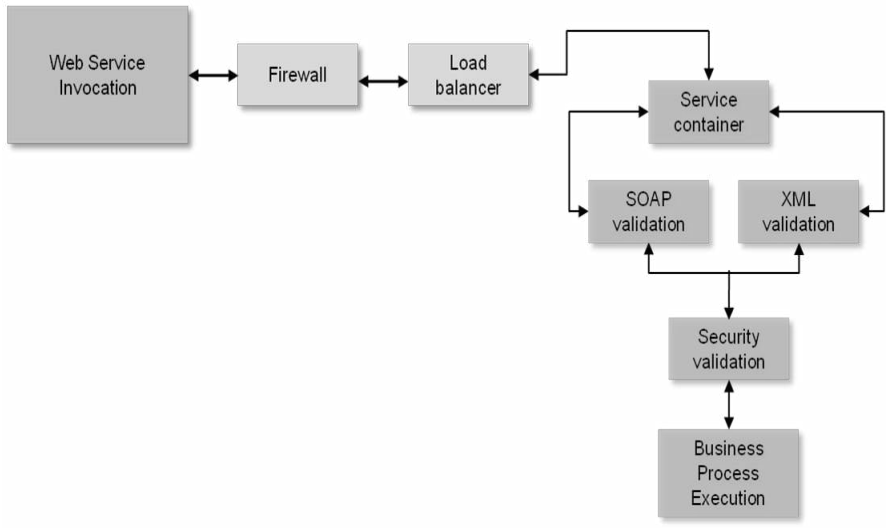
Although we have specifically called out interoperability and security, there are many other quality attributes that must be exhibited and , therefore, which need to be tested. The particular collection of quality attributes will depend on the organization's needs; we call out performance and reliability because they are commonly required and provide examples of testing for quality attributes.

7.1 Performance

Performance in a system is all about timing. Concerns include execution speed, latency, and the characteristics of event arrival.

As we discussed previously (Section 5.2.1), testing is allied with the evaluation process by which components were selected. Testing can thus be performed to ensure that products within the SOA infrastructure meet the performance criteria for which they should have been evaluated. Infrastructure components can be tested individually to determine actual values for both performance and latency, though these results will be of limited value unless the test environment closely mimics the operational environment. [18]

As is evident in Figure 5, there are multiple areas where performance can be compromised in the invocation of even a single web service, making it difficult to establish performance levels on infrastructure components like parsers and business process execution engines. This difficulty arises from the dynamic nature of web service invocation and the effect it has on attributes such as payload, process complexity, and dependence on other infrastructure components [44]. Performance testing can also include load and stress testing and can be the source of data for fine tuning to mitigate and address some performance-related bottlenecks.



[18] Performance requirements should specify environment context. Load and boundary conditions are critical to proper evaluation of performance.

Figure 5: Potential Performance Bottleneck Points in a Web Service Invocation

In addition, perspective is important for performance testing, because the testing can be conducted from the service container and service invoker viewpoints. As an example, Table 5 shows the differences in viewing throughput and latency from the service container and service invoker perspectives.

Table 5: Performance Metrics

	Service Container	**Service Invoker**
Throughput	Typically measured in terms of requests/sec.	The average data rate, including latency; usually measured in data rate/sec.
Latency	Time between service request and service response	Time taken between service invocation and the service response

Even so, testing can generate potentially useful measures for the SOA infrastructure. Clearly performance of the infrastructure will affect **every** thread, and testing can provide insight into the behavior of end-to-end threads. Testing can also be used to tune the infrastructure, by providing accurate measures of response time and latency that can be used as data for making engineering decisions such as the size of a cache or even which of a number of alternate algorithms performs the best. Infrastructure testing should measure not only overall response time but also time consumed by individual components or services in order to identify correctly the cause of poor responsiveness.

Recommendations for SOA Testing

Rec. 46: Test response time and latency for SOA infrastructure components and component interactions.

Rec. 47: Capture performance data both from the standpoint of the service container and the service invoker, in realistic scenarios and mission threads, and under varying service, SOA infrastructure, and network loads.

Both infrastructure services (such as service lookup, data storage, and data translation) and business services can be tested to determine whether they meet performance targets in the test environment. The assumption underlying this testing is that if they fail to meet their performance targets in a laboratory-like environment it is unlikely that they will do so in a production environment. A service will likely behave differently in a realistic setting rather than in a laboratory one (due to environmental factors such as server loads, bandwidth limitations, and network latencies will skew the results). One advantage of such tests, though, is that they may be able to be used to distinguish between different implementations of the same service. Overall, however, the best way to establish the adequacy of the performance of a web service is to consider it in the context of the scenarios and mission threads in which it executes.

Evaluation, verification, and monitoring of performance properties of web services involve activities similar to those for other distributed computing technologies. White box testing can be performed when implementation code is available, but in many cases, services will need to be treated as if they are black boxes. Similarly, though performance modeling of services is not yet a mainstream practice, it is recommended for the cases where there is sufficient insight into the service to construct the model.

The effect of unpredictable loads is greater when services are shared or are composite services. For example, if a service has multiple consumers, the increase in load from one consumer will likely degrade performance to all other consumers. In the case of underperforming composite services, it may not be clear whether the lack of performance is due to load on the service or network or because of a performance bottleneck in one of the services the composite invokes. For composite services, an additional difficulty is that the specific elements of a composite service may not be known until runtime. This challenge for testing is a result of an SOA benefit—the binding between the invoking service and the implementation of the services invoked can occur at runtime.

There are several testing tools (some are identified in Section 9) that provide detailed performance testing for web services. These tools not only test web service performance, but also provide performance testing capabilities for database connections, service registries, messaging infrastructure, and the like. They also provide the ability to simulate number/types of users and offer distributed load testing.

Recommendations for SOA Testing

Rec. 48: Establish the adequacy of the performance of a web service by considering it in the context of the scenarios and mission threads in which it executes.

Rec. 49: Where possible use models that simulate the service as a basis for performance modeling.

7.2 Reliability

Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment. The overall reliability of any software component cannot be adequately tested without a proper understanding of its fault model—the model of what can go wrong and from which the consequences of a fault can be predicted. This is certainly the case with individual services, and the overall reliability of an application cannot be tested without a proper understanding of the fault models of underlying services [45].

However, in most cases this fault model is hidden from the service-oriented application developer. As a result, the developer often finds it difficult to design tests to stress boundary conditions of the service through fault injection or other techniques. In addition, it becomes difficult for the application developer to find the root causes of problems encountered during reliability testing, whether the service will recover from a failure, or if the implementation has crashed or just failed because of specific input.

Another factor that makes evaluation of reliability difficult is the frequent lack of a central coordinator to direct and monitor complex transactions for the application. Participating web services have the option of whether to implement an internal transaction mechanism. If a transaction mechanism is not provided, managing transactions becomes the responsibility of the application. This can make it difficult to assure transactional integrity.

For example, consider an airline ticket booking web service. By itself, the airline booking service does not need to be transactional—it either books a ticket or not. However, if an application developer intends to provide a vacation booking capability by combining airline ticket booking with

train ticketing and car and hotel booking services, the entire vacation booking capability must be transactional. A transaction will be considered complete only after all the bookings have been confirmed. If each individual service is non-transactional, the responsibility is put on the application developer to implement transactions. This complicates implementation and testing of composite functionality. In the above example, the airline ticket, car, and hotel booking services would need to have a compensating operation to cancel reservations to be called in these cases.

Recommendations for SOA Testing

Rec. 50: *Establish a policy for publishing fault models for individual services and service compositions. The fault models should address common situations such as invalid input or internal data values, timing conditions (e.g., deadlock, race, and concurrency).*

Rec. 51: *Develop recommendations for the use of fault models to generate test cases.*

8 Testing for Standards Conformance

SOA implementations depend on the use of standards: Even when the infrastructure is wholly proprietary, it sets standards to which all services and service consumers must conform. Most implementations, though, are based on web services, and a critical factor in the success of SOA implementations has been the widespread acceptance of standards related to web services and RESTful services. Testing for compliance to the appropriate standards is necessary to ensure that basic expectations are met for participation in SOA applications.

The use of even common web service standards (e.g., HTML, XML, WSDL, and SOAP) does not guarantee interoperability because of vendors' customized implementation of the standards. For this reason, the Web Service–Interoperability (WS-I) [6] profiles were developed as a set of non-proprietary web service specifications which—along with clarifications, refinements, interpretations and amplifications of those specifications—promote interoperability. Each web service implementation, including COTS implementations, should be checked against WS-I profiles to test for syntactic interoperability. Note that WS-I also provides a set of testing tools to validate the conformance to the recommended profiles. Some tools have the added ability to validate conformance to key WS-* standards and assist in viewing test data from web services that use Secure Sockets Layer (SSL) for encryption. Finally, most organizations have standards over and above those defined by web services and conformance to these standards must also be confirmed through testing.

Web services are based on open standards, some of which focus on QoS attributes such as security, federation, trust, notification, reliability, transactions, and business activities. In addition, there are specific standards for some architecturally significant attributes—such as SAML, for some architecturally significant attributes; SAML is used in security and identity contexts [46]. In this section, we focus on web services that implement the SOAP protocol over HTTP, with references to the corresponding technology/standard/protocol for REST where relevant.

Testing a web service and its associated definition for conformance to key standards or protocols involves steps in the following areas:

- *Testing and validating the service contract (applicable standard, WSDL).* The following steps are needed to validate the service contract of a web service:
  - WSDL is well formed.
  - Implementation conforms to the appropriate WS-I profile.
  - Authentication standards are followed.
  - The web service conforms to the SOAP protocol.
  - The endpoints published in the WSDL are invoked using the web service deployed in the testing infrastructure.
  - Where appropriate, the inclusion of web service authentication compliance is checked for Basic, WS-Security, WS-Digest, and the like.
  - The WSDL types (what must be communicated) are validated using schemas.
  - The WSDL document is validated against WS-I Basic Profile 1.0 that provides interoperability guidance for a core set of non-proprietary web services specifications, such as

SOAP, WSDL, and UDDI, along with interoperability-promoting clarifications and amendments to those specifications.

- *Validating web service binding and message*. The web service is tested for conformance to the SOAP protocol over HTTP. The following SOAP-over-HTTP patterns are checked:
  - Request-Response Message Exchange Pattern
  - Response Message Exchange Pattern
  - SOAP Web Method Feature
  - SOAP Action Feature
  - SOAP binding (validated against the Simple SOAP Binding Profile 1.0 specification from the WS-I)
  - SOAP attachments (can be validated using W3C's SOAP Messages with Attachments standard)
  - SOAP over HTTPS (needs to be validated in cases where web services require data and transport security—i.e., the SSL certificate must be checked for validity)
- *Validating web service interoperability*. Compliance with standards provides confidence that the web service meets minimal levels of interoperability. The web service is tested for compliance interoperability by ensuring compliance with the following WS-I standards:
  - WS-I Basic Profile 1.0
  - Simple SOAP Binding Profile 1.0
  - W3C's SOAP Messages with Attachments standard
  - UDDI version 2.0 XML Schema standard

For compositions of web services, testing for conformance is more problematic. WS-BPEL is gaining widespread acceptance for web service orchestrations, and several tools exist to parse it and flag non-conforming use. However, it is possible (and perhaps more common) to build orchestrations with no WS-BPEL specification. Thus, conformance checking becomes difficult. For choreographies, the situation is even less clear. WS-CDL (a W3C product) does not appear to be widely used, suggesting that most choreographies being built have no specification that can be machine-parsed and validated for conformance [47].

Recommendations for SOA Testing

Rec. 52: Develop an automated strategy for conformance checking of web services, and integrate that strategy into governance processes.

Rec. 53: Develop automated tools to test for enterprise or organizational-unique conformance requirements as possible.

Rec. 54: Develop a strategy for conformance testing of service compositions.

Rec. 55: Check each web service implementation, including COTS implementations, against the WS-I profiles to test for syntactic interoperability.

9 Test-Related Strategies

In this section, we discuss other strategies that can be used in conjunction with testing to achieve confidence in the SOA system.

- Test-driven development and design-by-contract are strategies that developers can use to reduce the burden on testing through better development.
- Effective SOA governance can create policies that, when enforced, simplify the testing process.
- Runtime monitoring can be used to provide data that is used to tune the operational system. However, that same data can provide testers with valuable insights and assist them in tuning test cases to match better the reality of the operational system. In addition, performance requirements captured in SLAs can be dynamically tested through the use of runtime monitoring.
- Another strategy, assurance cases, can be used to determine what testing is necessary and, potentially, make regression testing more efficient by limiting the amount of retesting that has to be done.

9.1 Test-Driven Development

Test-Driven Development (TDD) constructs test cases prior to implementing a capability and then verifies the capability through the test cases. TDD practice encourages the development of test cases for small code increments and short iteration cycles. The quality of web services can be improved by using TDD [48]. Service developers can adopt a TDD approach to unit testing web services [49][50] during service development, before the services are deployed to the test or product environment. In a TDD approach, web services can be tested on a locally running server or deployed in a separate test environment. Also, integrated development environments provide mechanisms to perform unit testing of web services. For example, Microsoft Visual Studio .NET allows the creation of unit tests for web services that can be invoked either on a locally running web server or an active web server.

During TDD, service developers can test the service for correctness, path coverage, and response to bad inputs, among other aspects. However, since most testing is performed on non-production environments, it will not be possible to completely test quality attributes.

Provost discusses three different options for testing web services [49]. The first option uses class TDD techniques to test directly against the web services interface, similar to how clients of a web service will invoke it. The second option creates and tests business classes that get called from lightweight web services. The third option builds a custom XML grammar that allows creation of new tests without modifying original source code.

The use of TDD testing more effectively enables the integration of the testing process with development processes and governance. Ultimately, it may be possible to certify TDD processes such that a reduced level of post-development verification and validation (V&V) and certification ac-

tivity is necessary. This can lead to better code quality as well as significant reductions in the cumulative times for development, testing, V&V, and certification activities.

An obvious limitation of TDD is that it only applies to new development; if the SOA strategy is to create services that are merely wrappers to existing systems, it is too late for TDD to achieve its full potential. In these cases, testing as described throughout this report will need to be performed.

Recommendation for SOA Testing

Rec. 56: Devise a development strategy that includes TDD.

9.2 Design-by-Contract

An emerging paradigm that may ultimately be applied in SOA environments is the design-by-contract approach. Using design-by-contract tools, developers embed contract assertions in the form of source code comments. These contract assertions are then translated to instrument code assertions by the complier. When a contract violation occurs, the instrumentation code creates a notification that can be acted upon by an infrastructure component or by the invoking code. The concept behind design-by-contract has been applied in the past with varying degrees of success. It is out of scope for this report, though, to understand why it has failed to achieve its full potential.

Recommendation for SOA Testing

Rec. 57: Investigate the feasibility of design–by-contract approaches.

9.3 Governance Enforcement

Governance, which we characterize as developing and enforcing policy, provides the ability to manage, control, monitor, and enforce best policies and practices in an SOA deployment.

Governance applied at design time typically tests for compliance with a variety of standards and best practices (e.g., for compliance with Security Technology Implementation Guides [51] and coding standards). This governance should also require checks and validation of mandatory artifacts and metadata, so that services should be certifiable as compliant with the service on-boarding process. Such governance also leads to the provision of hooks and touch points to generate reports that provide information on whether a service has provided the required artifacts and whether these artifacts are valid and relevant in the desired context.

In cases where organizations group services into portfolios based on common capability or functionality, design time governance testing can ensure that the services assigned to a particular capability family comply with the mission threads and capabilities that constitute a specific capability.

Governance may also be applied at runtime, ensuring that a web service complies with the runtime policies and provides the tools and technologies to monitor the following service life-cycle events:

- service registration
- service discovery

- service delivery
- SLA enforcement
- security policy enforcement
- service availability
- service versioning
- service logging and auditing

The application of such governance serves as a valuable aid in recognizing unanticipated use scenarios and how these scenarios impact service orchestrations, choreography, and security policies. It also provides a "ring-side view" of service execution in an SOA environment and notifications when performance, security, and infrastructure thresholds are triggered. Governance can also apply to runtime monitoring that can shed light on service invocations that can compromise service availability and can better prepare service developers and infrastructure providers for scalability and availability.

Runtime data can provide a valuable insight for audit logs and for debugging service exceptions. The data can be used to review specific mission threads or use cases that trigger exceptions in certain services or when unanticipated scenarios cause a set of services to be dynamically orchestrated in a way that violates a security or network policy.

Recommendation for SOA Testing

Rec. 58: Runtime monitoring, and particularly runtime governance testing, should be used to evaluate the effect of unanticipated use scenarios on security and other QoS attributes of the SOA environment.

9.4 Runtime Monitoring

The category of tools and technologies that provide runtime monitoring capabilities is called Enterprise Service Management (ESM). This technology can be perceived as the *situational awareness enabler* in an SOA environment, with the ability to monitor runtime SOA activities and trigger events and notifications when policy is violated or when a software component ceases to execute. Some of the core capabilities of ESM technology are to

- monitor the execution of web services and infrastructure components in an SOA environment
- flag web services that may violate performance or latency thresholds
- support systems management tools with up-to-date runtime metrics on SOA infrastructure components and web services
- capture runtime data and generate reports that can be used to resolve disputes related to SLAs between service providers and service consumers
- monitor resource consumption for SOA infrastructure components, web services, and applications
- enforce security policies related to access and binding in the SOA environment
- enforce governance policies in the SOA environment

Because ESM technology is responsible for constant runtime monitoring of SOA infrastructure, it is deployed at key network gateways or as software agents on the SOA infrastructure to monitor interactions between web services and SOA infrastructure components (see Figure 6).

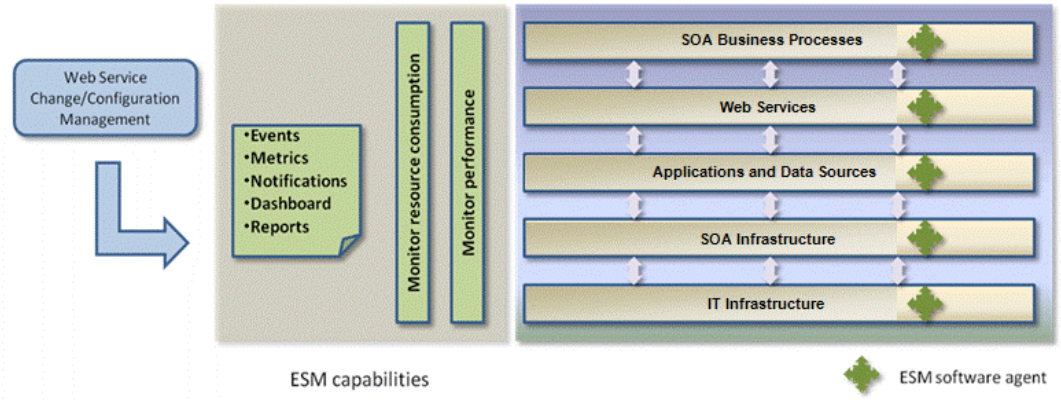


Figure 6: Enterprise Service Management (ESM) in an SOA Environment

Additionally, and importantly, the SOA infrastructure can have the capability to monitor performance and detect bottlenecks that may be degrading performance. Metrics can be collected and analyzed as a self-testing capability.

Recommendations for SOA Testing

Rec. 59: Develop a strategy for using runtime monitoring to supplement testing and capture data for analysis and improvement of the SOA infrastructures and web services.

Rec. 60: Runtime monitoring of performance should be used to detect peak performance degradation or services or components that are bottlenecks.

9.5 Service Level Agreements

SLAs are covenants between a service provider and service consumers that help to obtain a level of verifiable QoS. They play an import role in SOA because they

- provide a standard for service providers to advertise QoS attributes for web services
- allow runtime monitoring technologies in an SOA environment to monitor and report conformance to web service SLAs
- allow service consumers to select services based on SLA requirements, especially in scenarios where there are multiple service providers offering the same capability

From a testing perspective, two aspects need to be considered. One consideration is that service consumers need to specify their QoS requirements in a consistent manner. The second is that service providers need to match the needs of these consumers by advertising that their services' SLAs can be enforced through the service discovery, binding, and invocation life cycle of the infrastructure. For testing, service providers collect measurements for comparison to the metrics specified in the SLA. When SLA violations are detected, appropriate actions are taken such as dynamic reconfiguration of a service or selection of a different service [52].

There are two key standards for specifying SLAs in a web services environment:

- the Web Service Level Agreement (WSLA) specification, proposed by IBM [53]
- the Web Services Agreement Specification (WS-Agreement) specification, proposed by the Open Grid Forum (OGF) [54]

Based on recent trends, the WS-Agreement specification has seen broader adoption. IBM, BEA, Microsoft, SAP, Sonic, and VeriSign are working on WS-Policy and WS-PolicyAttachment, two specifications that address the quality of service aspects of web services SLAs. WS-Policy provides a specification by which service providers and consumers can use WS-Policy based artifacts to advertise and consume SLA policies. WS-PolicyAttachment specifically deals with the issue of attaching policy assertions to web services SLA policies.

Recommendations for SOA Testing

Rec. 61: Whenever possible, specify SLAs in a machine-readable format that will allow automatic monitoring of agreements at runtime.

Rec. 62: Develop monitoring capabilities that can collect and log data relevant to SLAs, and identify SLA violations when possible

Rec. 63: Develop a strategy for service consumers and providers to provide QoS information in a consistent and standards-driven manner.

Rec. 64: Develop a strategy for SLA creation and validation, oriented toward commercial standards where possible.

9.6 Assurance Cases

The National Research Council (NRC), in a report entitled *Software for Dependable Systems: Sufficient Evidence?* [55], finds that even large-scale, end-to-end testing says little about dependability for systems where extraordinarily high levels of dependability are required. Similar statements can be made about other quality attributes such as security and performance: even extensive end-to-end testing can only demonstrate that there are some conditions under which the systems of systems overall will do something expected and useful. It is risky to extrapolate to broader conclusions about overall system-of-systems behavior on the basis of a set of end-to-end test cases.

For testing to be a credible indicator of overall system-of-systems qualities, the relationship between testing and the properties being claimed must be explicitly justified. The NRC report recommends a well-reasoned argument based upon evidence such as formal proofs, static code checking, test results, simulation results, and so on.

The SEI is applying the assurance case method to the analysis of end-to-end threads in SOA implementations. The purpose of assurance cases is to provide confidence that a capability will perform as expected. An assurance case is similar in nature to a legal case in that it links specific claims to evidence relevant to the claim and to arguments as to what the evidence supports or does not support. In an SOA context, evidence (potentially from a test) is linked with arguments about why the evidence supports specific claims for the SOA implementation (e.g., overall performance). Thus, assurance cases can be used to identify the scope of testing after a change has been made. The uniting of evidence and the argument mitigates the risk of extrapolating from end-to-end test cases to overall claims about the SOA implementation.

Assurance cases can augment testing because the case identifies the evidence tests need to provide and the significance of that evidence in supporting claims about overall system behavior. Further, assurance cases can help determine whether test results show what they purport to show and whether the appropriate things have been tested.

Information is available from the SEI about the applicability of assurance cases to dependability [56], security [57], and survivability [58].

Recommendation for SOA Testing

Rec. 65: Develop a strategy that supports the development of well-reasoned arguments about the quality of testing for critical mission threads.

10 Summary

Testing SOA implementations is challenging for various reasons. For example, services are often outside the control of the organization consuming the service (i.e., service developers are independent from service providers and service consumers), leading to potential mismatches and misunderstandings between parties. Also, SOA implementations are often highly dynamic, with frequently changing services and service consumers and varying load on services, SOA infrastructure, and the underlying network. Consequently, it is normally not possible to replicate all possible configurations and loads during the testing process. What is possible is to identify and maintain a set of configurations that are deemed important and use them as the basis for defining the test environments.

We suggest in this report that testing SOA implementations needs to account for elements that reflect functional, non-functional (as perceived through the quality attributes of the services), and conformance aspects. By devising a strategy for testing the SOA infrastructure, web services (individual and composite services), and end-to-end threads, those responsible for testing the SOA implementation can more comprehensively perform unit, integration, and system-level testing (also known as dynamic testing). Aiding that effort, which is bound to be fraught with many challenges, are tools that can automate testing and web service verification processes.

In discussing the aspects, elements, and processes associated with SOA implementation testing, we make many recommendations, each phrased in the form of an assertion (a "do" statement). Our recommendations stress a few fundamental principles, such as:

- It is important to view testing in a strategic way, recognizing that the SOA implementation is meant to serve business and mission goals.
- A testing strategy should not ignore the complexity of SOA implementations that can arise from the absence of full information about services and interfaces, rapid service upgrade cycles, and other factors.
- A testing strategy is incomplete if it leaves out the testing of end-to-end threads.

In addition to making recommendations for solid practice in SOA implementation testing, this report also points to areas where further examination is warranted. In particular, the use of tools to automate testing and the development of more efficient web services verification processes stand out. Automation in testing, for instance, can make regression testing timelier, which can more readily meet the needs brought about by the frequency of service updates and the implications for other consumers of a change to any one service. Efficient web service verification, while filling security and information assurance needs, can allow new and upgraded services to be incorporated with confidence in less time. This can allow services to be made available with the same agility in which they are developed.

Appendix A List of Acronyms Used

Acronym	Description
ATO	Authority to Operate
BPEL	Business Process Execution Language
CIO/G6	Chief Information Officer/G6
COTS	Commercial off-the-Shelf
ESB	Enterprise Service Bus
ESM	Enterprise Service Management
GML	Geography Markup Language
GOTS	Government Off-The-Shelf
HTML	Hyper Text Markup Language
HTTP	Hypertext Transport Protocol
HTTPS	Hyper Text Transport Protocol Secure
NIST	National Institute of Standards and Technology
NRC	National Research Council
OASIS	Organization for the Advancement of Structured Information Standards
OSI	Open Systems Interconnection
OSIRM	Open Systems Interconnection (OSI) Basic Reference Model
OWL	Ontology Web Language
OWL-S	Formerly DAML-S; is the Ontology Web Language for Services
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
RTSS	Research, Technology, and System Solutions (RTSS)
SAML	Security Assertion Markup Language
SEI	Software Engineering Institute
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TDD	Test-Driven Development
TST	Time Sensitive Targeting
UCORE	Universal Core

UDDI	Universal Description Discovery and Integration
V&V	Verification and validation
W3C	World Wide Web Consortium
WS-I	Web Services Interoperability Organization
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
WS-BPEL	Web Services Business Process Execution Language
XML	eXtensible Markup Language

Appendix B Consolidated List of Recommendations

Rec. 1: *Develop an SOA test strategy that accounts for SOA infrastructure elements, web services (individual and composites), and end-to-end threads.*

Rec. 2: *Establish guidelines for testing the service before it is ready for use either by the service consumers or the service integrator.*

Rec. 3: *Establish clear lines of testing and customer support responsibility that allow for distinctions between the roles of service developers and service providers.*

Rec. 4: *Develop governance processes that support service consumers in the identification of use cases and the tests necessary to achieve appropriate assurance about performance of the service within those use cases.*

Rec. 5: *Develop guidelines for testing composites of various types, including composites that implement capabilities supporting pre-defined mission threads and ad hoc composites that reflect attempts to address emerging needs.*

Rec. 6: *Develop guidelines for testing composites employing the range of composition mechanisms expected (e.g., WS-BPEL defined, application-embedded composites, WS-CDL)*

Rec. 7: *Develop guidelines and governance processes for testing/verification of new and revised infrastructure capabilities, including notification of users of infrastructure changes and triggers for retesting.*

Rec. 8: *Develop policies for the type and level of testing support provided by the infrastructure provider to the service provider or integrator.*

Rec. 9: *Identify the focus, expectations, and limitations of third-party testing/certification activities.*

Rec. 10: *Develop policy clearly delineating the forms of testing performed according to SOA roles.*

Rec. 11: *Provide developers with a fully outfitted test environment for early testing of services and service orchestrations; provide this environment to testing organizations, too, for performing certification activities.*

Rec. 12: *Develop strategies for testing of the service registration mechanism.*

Rec. 13: *Develop tests to determine whether authorized service consumers can find services based on syntactic or semantic approaches.*

Rec. 14: *Develop tests that reveal whether all appropriate service providers and consumers are notified when a new version of a service is registered with the infrastructure or when service levels are violated.*

Rec. 15: *Develop a testing strategy for service monitoring capabilities.*

Rec. 16: *Develop a testing strategy for service virtualization features.*

Rec. 17: *Provide support for service mocking, including mocking of services on remote platforms and infrastructures.*

Rec. 18: *Integrate service mocking into the development and testing strategies.*

Rec. 19: *Develop guidelines for the type of faults that services and service compositions must be tested for.*

Rec. 20: *Formal agreements should be in place with data providers to identify test data sets and test interfaces.*

Rec. 21: Establish baselines for services, compositions, and end-to-end threads.

Rec. 22: Establish regression testing triggers for each.

Rec. 23: Establish a regression testing sandbox fitted with appropriate testing and analysis tools to automate regression testing to the degree possible.

Rec. 24: Use approaches such as assurance cases to identify when additional regression testing is needed.

Rec. 25: Develop a strategy for testing at each of the layers of the SOA interoperation stack.

Rec. 26: Instrument test instances to help identify the sources of failures.

Rec. 27: Perform regression testing to identify the impacts of changes on services, nodes and components along the pathway that is exercised.

Rec. 28: Develop a strategy for identifying the important threads in an end-to-end threads and decorating those threads with both functional and QoS expectations.

Rec. 29: Develop test cases for critical threads that have been identified.

Rec. 30 For certain types of services (e.g., public web services on the internet) where the actual operating context (e.g., expected load and usage, level of security required, etc.) is unknown until the service is actually being used, it may be necessary to adopt an incremental approach in which functionality is fielded and the system is then adjusted to meet QoS goals.

Rec. 31: Develop a strategy for identifying and testing likely dynamically composed threads.

Rec. 32: Implement "non-interference" strategies such that other threads cannot compromise functioning or QoS in the rest of the end-to-end thread..

Rec. 33: Develop a strategy for cross-site and cross-organizational collaboration to identify test context and data loads.

Rec. 34: Develop a strategy for cross-site and cross-organizational verification of changes in back-end systems and capturing and analysis of test results.

Rec. 35: Develop a strategy to identify and test the capabilities of SOA infrastructures that are needed to federate the infrastructure.

Rec. 36: Develop a list of interoperability standards that services must comply with, and consider offering an interoperability compliance check as a service to developers.

Rec. 37: Develop a list of data models that services must use and consider offering a compliance check as a service to developers.

Rec. 38: Develop a strategy to supports scenario-driven testing of interoperability.

Rec. 39: Develop policies for differential treatment of predefined and dynamic compositions of services.

Rec. 40: Develop policies for capturing dynamically created threads for future analysis and inclusion into the collection of predefined threads.

Rec. 41: Develop guidelines for testing for security of SOA infrastructure, services it provides or hosts, and the environment in which it exists based on the level of security required.

Rec. 42: Consider the entire attack surface in the security testing of all SOA components.

Rec. 43: Develop strategies for testing of services where source code is available and for services for which source code is unavailable.

Rec. 44: Develop an integrated security strategy that considers the network, the SOA infrastructure, web services.

Rec. 45: Web service verification should be used with testing to gain confidence in the confidentiality, integrity, and availability of the SOA infrastructure, other web services, back-end systems, end-to-end mission threads, and critical data.

Rec. 46: Test response time and latency for SOA infrastructure components and component interactions.

Rec. 47: Capture performance data both from the standpoint of the service container and the service invoker, in realistic scenarios and mission threads, and under varying service, SOA infrastructure, and network loads.

Rec. 48: Establish the adequacy of the performance of a web service by considering it in the context of the scenarios and mission threads in which it executes.

Rec. 49: Where possible use models that simulate the service as a basis for performance modeling.

Rec. 50: Establish a policy for publishing fault models for individual services and service compositions. The fault models should address common situations such as invalid input or internal data values, timing conditions (e.g., deadlock, race, and concurrency).

Rec. 51: Develop recommendations for the use of fault models to generate test cases.

Rec. 52: Develop an automated strategy for conformance checking of web services, and integrate that strategy into governance processes.

Rec. 53: Develop automated tools to test for enterprise or organizational-unique conformance requirements as possible.

Rec. 54: Develop a strategy for conformance testing of service compositions.

Rec. 55: Check each web service implementation, including COTS implementations, against the WS-I profiles to test for syntactic interoperability.

Rec. 56: Devise a development strategy that includes TDD.

Rec. 57: Investigate the feasibility of design–by-contract approaches.

Rec. 58: Runtime monitoring, and particularly runtime governance testing, should be used to evaluate the effect of unanticipated use scenarios on security and other QoS attributes of the SOA environment.

Rec. 59: Develop a strategy for using runtime monitoring to supplement testing and capture data for analysis and improvement of the SOA infrastructures and web services.

Rec. 60: Runtime monitoring of performance should be used to detect peak performance degradation or services or components that are bottlenecks.

Rec. 61: Whenever possible, specify SLAs in a machine-readable format that will allow automatic monitoring of agreements at runtime.

Rec. 62: Develop monitoring capabilities that can collect and log data relevant to SLAs, and identify SLA violations when possible.

Rec. 63: Develop a strategy for service consumers and providers to provide QoS information in a consistent and standards-driven manner.

Rec. 64: Develop a strategy for SLA creation and validation, oriented toward commercial standards where possible.

Rec. 65: Develop a strategy that supports the development of well-reasoned arguments about the quality of testing for critical mission threads.

Appendix C Key Attributes for Testing Web Services

Web Service Inspection

1. SOAP 1.1 & 1.2 Support
2. WSDL Viewer and Validation
3. XML Schema Inspection
4. XML Table Inspection
5. Web Service Form Editor
6. Web Service Overview
7. SOAP Monitoring

Web Service Invocation

1. Automatic Request Generation
2. Endpoint Management
3. WS-Security Standard Support
4. WS-Attachment Support (MTOM, SOAP Attachment, Inline)
5. Custom HTTP header for REST
6. Raw Messages
7. Web Service Form Editor
8. Web Service Tree Editor
9. WS-Security Support
10. Web Service Recording
11. WSDL Validation

Functional Testing

1. WSDL Coverage
2. Request/Response Coverage
3. Message Assertion
4. Test Refactoring
5. Test Refactoring
6. Drag and Drop Test Creation
7. Message Pretty Printing
8. Coding Free Test Assertion
9. Running of Multiple Tests
10. Test Logs
11. Test Configuration
12. Easy Content Transfer
13. Data Source Driven Tests

Functional Testing, continued

1. Data Collection
2. MockResponse
3. Maven Integration
4. Standalone Server Runners
5. Scripting Support
6. Scripting Libraries
7. Requirements Management
8. Reporting
9. Integrated Reporting
10. Form Based input for easy manual testing
11. Tree Based input for easy manual testing
12. Create Test from Web Service Recordings
13. WS-Security Support
14. WS-I Integration
15. Web Service Recording

Load Testing

1. Rapid Load Tests from Functional Tests
2. Configurable Load Strategies
3. LoadTest Assertions
4. Real Time Statistics
5. Real Time Diagrams
6. Statistics Exporting
7. Performance Monitoring
8. Run from command line and Maven

Web Service Simulation: MockServices

1. Create Automatic MockService From WSDL
2. Add Operations to MockService
3. Create Custom Responses
4. Multiple Dispatching Options including Scripts
5. WS-Security Support
6. WSDL Publishing
7. SSL Support
8. WSDL Coverage
9. Run from command line and Maven

Web Service Development and Validation

1. Generate Server and Client Code
2. Generated XML Bindings
3. Command Line Support
4. Validate Web Service Definitions
5. Validate Request and Response Bodies
6. IDE Integration

Appendix D Testing Process Preconditions and Deliverables

Testing of an SOA implementation should begin early and occur often, yet should not discourage engineers from achieving the desired agility often cited as a primary goal of SOA. This represents a potential conflict between achieving agility and performing rigorous testing. Many organizations are addressing this conflict by employing multi-phased processes where individual services must overcome low entry barriers for initial, discovery-oriented, phases, followed by increasingly higher barriers in subsequent phases leading ultimately to widespread deployment.

Preconditions for Testing Process

We strongly endorse this phased strategy. Ultimately, however, formal testing for making deployment decisions must begin. We suggest the following information be made available to testers prior to the initiation of such testing. Subsets of this information may make up the core of the entry barriers for earlier testing phases:

For Web Services and Composites:

- Architectural and design documentation in accordance with the organization's standards and guidelines. Information about documenting software architectures can be found in *Documenting Software Architectures: Views and Beyond* [59].
- Service metadata: information about the provider, developer, versions and releases, technology dependencies, platforms, security classification/mission access categories, data and network access. This metadata should be incorporated into the SLA.
- WSDL specifications
- Service source code (if available)
- SLAs capturing
  - Information necessary to deploy the service: ports, protocols, scripts, guidance and constraints
  - Information necessary to access the service
  - Information about the semantics of the service: business rules, behaviors, use cases, and sequencing
  - Information about runtime dependencies: runtime environment and dependencies on other applications and data sources, etc.
  - Information about operations: contact points, change management, versions and deployment, support, fault reporting, etc.
  - Information about QoS provided (availability, response time, throughput, fault tolerance, etc.)
  - Information about security (policies, authority to operate (ATO) certification authorities, criteria and restrictions, vulnerabilities, monitoring, etc.)
  - Constraints on configuration, deployment, execution of the service
  - Information about testing that has been accomplished (type, parties, setup, tool support, results, etc.)

For SOA Infrastructure

- Information analogous to that required for web services, plus:
- Configuration information and data for the various infrastructure components
- Licenses and agreements with providers of individual capabilities

For end-to-end threads

- information analogous to that provided for web services
- metadata about the thread
- code controlling orchestration (e.g., BPEL for business process threads) of the thread, if available
- documents analogous to SLAs, but presented from the perspective of the mission thread
- documents capturing requirements and other expectations for the thread

Deliverables from the Testing Effort

The IEEE Standard for Software Test Documentation [60] provides a starting point for identifying deliverables for any testing effort. SOA-specific implementation guidance is as follows:

Documents related to test specifications

- Test design specifications identifying the approach, test procedures and cases, and pass/fail criteria. An SOA test design specification needs to clearly identify unique features of SOA testing that affect the testing approach, tasks, and environment. The testing approach should identify appropriate strategies for testing the SOA infrastructure, services with and without source code availability, SOA services deployed on "local" infrastructures and "remote" infrastructures, and end-to-end testing of predefined and dynamically composed mission threads. The desired characteristics of the testing environment and the responsibilities of all participants in that environment (e.g., loading of test data, capturing of metrics) should be clearly defined.
- Test case specifications identifying the input, output, contextual data, and constraints on procedures. SOA test case specifications should include flexibility to incorporate automated testing techniques such as fuzzing that produce multiple executions based on variations of input parameters and attachments. In addition, test case specifications should address desired quality of service attributes; pay particular attention to the hardware and software configurations; and identify appropriate network, SOA environment, and service loads at the time of testing.
- Test procedure specifications identifying steps to operate the environment and run the tests. SOA test procedures are likely to be complex due to the distributed and dynamic nature of SOA environments. There may be extensive procedures for establishing the environment context prior to testing (e.g., to load appropriate data in remote databases), for executing tests, and for capturing the results of the test. For service and end-to-end testing, verification of a test may require access to information on remote machines and back end data stores.

Documents related to test reporting:

- Test item transmittal reports identifying the communication mechanism between implementation and testing groups. The transmittal reports should reflect the range of involved parties (i.e., for SOA infrastructure, individual services, composites, end-to-end threads).
- Test log records of what occurred. Test logs should capture—to the extent possible—the exact environment context at the point of the test. This includes, loads on networks, SOA infrastructure and services identification and version records, and specific hardware and software configurations and versions. This information will be essential for analyzing test results and anomalous events that occur. Logs may also have to address issues such as lack of universal time across the distributed environment (i.e., inconsistent timestamps).
- Test incident reports that record incidents that require further investigation. Incident reports are often given inadequate attention even in traditional development. This leads to difficulty in isolating errors and reproducing problems. For SOA implementations, failure to capture complete descriptions of problems and the context in which they occur may be even more damaging due to the distributed nature of SOA applications. Test summary reports summarizing the testing activity. These reports may contain findings and recommendations about all elements of the SOA implementation.

References

URLs are valid as of the publication date of this document.

[1] G. A. Lewis, E. Morris, S. Simanta, and L. Wrage, "Common Misconceptions about Service-Oriented Architecture," in *Proceedings of the Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, Banff, Alberta, Canada, 2007, pp. 27-30.

[2] HTTP. (2003) World Wide Web Consortium. HTTP - Hypertext Transfer Protocol. [Online]. http://www.w3.org/Protocols/

[3] SOAP. (2003) World Wide Web Consortium. HTTP - Hypertext Transfer Protocol. [Online]. http://www.w3.org/Protocols/

[4] WSDL. (2005, Aug.) Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft. [Online]. http://www.w3.org/TR/wsdl20/

[5] UDDI . (2005) Organization for the Advancement of Structured Information Standards. OASIS UDDI. [Online]. http://www.uddi.org/

[6] C. Kaner, J. Falk, and H. Q. Nguyen, *Testing Computer Software*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1999.

[7] J. A. Whittaker, "What Is Software Testing? And Why Is It So Hard? ," *IEEE Software*, vol. 17, pp. 70-79, 2000.

[8] L. O'Brien, L. Bass, and P. Merson, "Quality Attributes and Service-Oriented Architectures," Technical Note CMU/SEI-2005-TN-014, 2005. [Online]. http://www.sei.cmu.edu/library/reports/abstracts/05tn014.cfm

[9] (2009) World Wide Web Consortium. [Online]. http://www.w3.org/

[10] Web Services Interoperability Organization. [Online]. http://www.ws-i.org/

[11] Organization for the Advancement of Structured Information Standards (OASIS). [Online]. http://www.oasis-open.org/home/index.php

[12] (JSAWG), Joint Security Architecture Working Group, "SOA Security Requirements, Working Group Report v 0.11- 1/14/08," 2008.

[13] F. Hueppi, L. Wrage, and G. A. Lewis, "T-Check in Technologies for Interoperability: Business Process Management in a Web Services Context," Technical Note CMU/SEI-2008-TN-005, 2008. [Online]. http://www.sei.cmu.edu/library/reports/abstracts/08tn005.cfm

[14] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, pp. 46-52, 2003.

[15] M. Rosen. (2008, Apr.) www.bptrends.com. [Online]. http://docs.google.com/gview?a=v&q=cache:QdG8l6xlZEwJ:www.bptrends.com/publicationfiles/04-08-COL-BPMandSOA-OrchestrationorChoreography-%25200804-Rosen%2520v01%2520_MR_final.doc.pdf+SOA+service+orchestration+verses+choreography&hl=en&gl=us&sig=AFQjCNG1eN

[16] R. A. Paul, "DoD towards software services," in *WORDS '05: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, 2005, pp. 3-6.

[17] K. C. Morris and D. Flater, "Standards-based Software Testing in a Net-Centric World," in *Proceedings of the Ninth International Workshop on Software Technology and Engineering Practice (STEP 1999)*, Pittsburgh, 1999, pp. 115-122.

[18] REST. (2010) Wikimedia Foundation. Representational State Transfer. [Online]. http://en.wikipedia.org/wiki/Representational_State_Transfer

[19] WS-Notificatino. (2010) Organization for the Advancement of Structured Information Standards. OASIS Web Services Notifi-cation (WSN) TC. [Online]. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[20] WS-Trust. (2007) Organization for the Advancement of Structured Information Standards. WS-Trust 1.3. OASIS Standard. [Online]. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html

[21] M. Penta, M. Bruno, G. Esposito, V. Mazza, and G. Canfora, "Web Services Regression Testing," in *Test and Analysis of Web Services*. Heidelberg: Springer Berlin, 2007.

[22] B. Woolf, "Streamline SOA Development using Service Mocks," in *developerWorks.* IBM, 2005, vol. 2009.

[23] U. Godage, "Mock Web services with Apache Synapse to develop and test Web services," in *developerWorks | SOA and Web services*. IBM, 2008, vol. 2009.

[24] B. Lublinsky, "Mocking Web Services," *InfoQ*, 2008.

[25] L. F. de Almeida and S. R. Vergilio, "Exploring Perturbation Based Testing for Web Services," in *International Conference on Web Services (ICWS '06)*, 2006, pp. 717-726.

[26] M. G. Fugini, B. Pernici, and F. Ramoni, "Quality Analysis of Composed Services through Fault Injection," in *Business Process Management Workshops.* Heidelberg: Springer Berlin, 2008, vol. 4928, pp. 245-256.

[27] X. Wuzhi, J. Offutt, and L. Juan, "Testing Web services by XML perturbation," in

Software Reliability Engineering: Proceedings of the 16th IEEE International Symposium (ISSRE 2005), Chicago, 2005, pp. 257-266.

[28] Y. Sikri, "End-to-End Testing for SOA-Based Systems," *MSDN Architecture Center*, vol. 2008, 2007.

[29] M. Gagliard and W. Wood, "System of Systems Architecture Evaluation with Concurrent Development," in *Third SEI Software Architecture Technology User Network Workshop (SATURN 2007)*, Pittsburgh, 2007.

[30] X. Bai, C. P. Lam, and H. Li, "An Approach to Generate the Thin-Threads from the UML Diagrams," in *28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, 2004, pp. 546-552.

[31] N. Raza, A. Nadeem, and M. Z. Iqbal, "An Automated Approach to System Testing Based on Scenarios and Operations Contracts," in *Seventh International Conference on Quality Software (QSIC 2007)*, 2007, pp. 256-261.

[32] W. T. Tsai, C. Fan, Z. Cao, B. Xiao, and H. Huang, "A Scenario-Based Service-Oriented Rapid Multi-Agent Distributed Modeling and Simulation Framework for SoS/SOA and Its Applications," in *Foundations '04: A Workshop for VV&A in the 21st Century*, 2004.

[33] WS-BPEL. (2007) Organization for the Advancement of Structured Information Standards. Web Services Business Process Execution Standard Version 2.0 . [Online]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[34] Department of Homeland Security. National Vulnerability Database. [Online]. http://nvd.nist.gov/

[35] Microsoft Corporation. (2005) [Online]. http://msdn.microsoft.com/en-us/library/ms954176.aspx

[36] OASIS. (2005, Feb.) [Online]. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[37] M. Howard, "Fending Off Future Attacks by Reducing Attack Surface," 2003.

[38] P. K. Manadhata and J. M. Wing, "Measuring a System's Attack Surface," Carnegie Mellon University, Technical Report CMU-CS-04-102, 2004.

[39] P. K. Manadhata, K. Tan, R. A. Maxion, and J. M. Wing, "An Approach to Measuring A System's Attack Surface," Carnegie Mellon University CMU-CS-07-146, 2007. [Online]. http://reports-archive.adm.cs.cmu.edu/anon/2007/CMU-CS-07-146.pdf

[40] M. Hafner and R. Breu, *Security Engineering for Service-Oriented Architectures*. Berlin, Germany: Springer, 2009.

[41] A. Singhal, T. Winograd, and K. Scarefone. (2007, Aug.) Guide to Secure Web Services. [Online]. http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf

[42] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh. (2008, Sep.) Technical Guide to Information Security Testing and Assessment. [Online]. http://csrc.nist.gov/publications/nistpubs/800-115/SP800-115.pdf

[43] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, and L. Martino, "Challenges of Testing Web Services and Security in SOA Implementations," in *Test and Analysis of Web Services*. Heidelberg: Springer Berlin, 2007, pp. 395-440.

[44] I. Crnkovic, M. Larsson, and O. Preiss, "Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes," *Architecting Dependable Systems*, vol. III, pp. 257-278, 2005.

[45] V. Cortellessa and V. Grassi, "Reliability Modeling and Analysis of Service-Oriented Architectures," in *Test and Analysis of Web Services*. Heidelberg: Springer Berlin, 2007, pp. 339-362.

[46] SAML. (2010) Wikimedia Foundation, Security Assertion Markup Language. [Online]. http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language

[47] WS-CDL. (2004) Worldwide Web Consortium. Web Services Choreography Description Language Version 1.0. [Online]. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/

[48] K. Beck, *Test Driven Development: By Example*. Addison-Wesley Professional, , 2002.

[49] P. Provost, "Test-Driven Development and Web Services," *Geek Noise*, 2009.

[50] D. Vines, "Test-driven development in an SOA environment: Part 1: Testing data maps," in *developerWorks WebSphere | SOA and Web Services*. IBM, 2008., vol. 2009.

[51] Security Technology Implementation Guides. [Online]. http://iase.disa.mil/stigs/index.html

[52] P. Bianco, G. A. Lewis, and P. Merson, "Service Level Agreements in Service-Oriented Architecture Environment," Technical Note CMU/SEI-2008-TN-021, 2008. [Online]. http://www.sei.cmu.edu/library/abstracts/reports/08tn021.cfm

[53] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification, Version 1.0," 2003. [Online]. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[54] A. Andrieux, et al., "Web Services Agreement Specification (WS-Agreement)," 2007. [Online]. http://www.ogf.org/documents/GFD.107.pdf

[55] D. Jackson, M. Thomas, and L. I. Millett, *Software for Dependable Systems: Sufficient*

Evidence?. The National Academy Press, 2007.

[56] J. Goodenough, C. Weinstock, and J. J. Hudak, "Dependability Cases," Technical Note CMU/SEI-2004-TN-016, 2004. [Online]. http://www.sei.cmu.edu/library/abstracts/reports/08tn016.cfm

[57] J. Goodenough, H. Lipson, and C. Weinstock. (2008) Build Security IN. [Online]. https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/assurance/643-BSI.html

[58] R. Ellison, J. Goodenough, C. Weinstock, and C. Woody, "Survivability Assurance for System of Systems," Pittsburgh, Technical Report CMU/SEI-2008-TR-008, 2008. [Online]. http://www.sei.cmu.edu/library/abstracts/reports/08tr008.cfm

[59] P. Clements, et al., *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2002.

[60] IEEE, *IEEE Standard for Software Test Documentation. vol. 829*. New York: IEEE Computer Society, 1998.

[61] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, pp. 267-284, 1999.

[62] P. Donham. (2007) Aberdeen Group Company Web Site. [Online]. http://www.aberdeen.com/summary/report/benchmark/4117-RA-soa-web-services.asp

[63] Software AG. (2008) Software AG web site. [Online]. http://www.softwareag.com/Corporate/res/

[64] L. F. a. V. S. R. s. l. de Almeida, "Exploring Perturbation Based Testing for Web Services," in *International Conference on Web Services (ICWS '06)*, 2006, pp. 717-726.

[65] P. K. Manadhata, K. Tan, R. A. Maxion, and J. M. Wing, "An Approach to Measuring A System's Attack Surface (CMU-CS-07-146)," Pittsburgh, Technical Report, 2007.

[66] G. Canfora and M. Di Penta, "Testing services and service-centric systems: challenges and opportunities," *IT Professional*, vol. 8, pp. 10-17, 2006.

[67] K. M. Kumar, A. S. Das, and S. Padmanabhuni, "WS-I Basic Profile: a practitioner's view," in *Proceedings of the IEEE International Conference on Web Services*, 2004, pp. 17-24.

[68] G. A. Lewis, E. Morris, S. Simanta, and L. Wrage, "Why Standards Are Not Enough to Guarantee End-to-End Interoperability," in *Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, 2008, pp. 164-173.

[69] J. Grundy, J. Hosking, L. Li, and N. Liu, "Performance engineering of service compositions," in *Proceedings of the 2006 international workshop on Service-oriented*

software engineering, 2006, pp. 26-32.

[70] M. R. Barbacci, et al., "Quality Attribute Workshops (QAWs), Third Edition," Pittsburgh, Technical Report CMU/SEI-2003-TR-016, 2003. [Online]. http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	March 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Testing in Service-Oriented Environments	FA8721-05-C-0003

6. AUTHOR(S)

Ed Morris, William Anderson, Sriram Bala, David Carney, John Morley, Patrick Place, & Soumya Simanta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TR-011

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2010-011

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This report makes recommendations for testing service-oriented architecture (SOA) implementations consisting of infrastructure, services, and end-to-end processes. Testing implementations of SOA infrastructure, services, and end-to-end processing in support of business processes is complex. SOA infrastructure is often composed of multiple, independently constructed commercial products—often from different vendors—that must be carefully configured to interact in an appropriate manner. Services are loosely coupled components which are intended to make minimal assumptions about where, why, and under what environmental conditions they are invoked. Business processes link together multiple services and other systems in support of specific tasks. These services and systems may operate on remote platforms controlled by different organizations and with different SOA infrastructures. Such complications make it difficult to establish appropriate environments for tests, to ensure specific qualities of service, and to keep testing up-to-date with changing configurations of platforms, infrastructure, services, and other components.

14. SUBJECT TERMS	15. NUMBER OF PAGES
SOA, service-oriented systems, service oriented architecture, web services, service testing	78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL