

# Key Strategies for SOA Testing

Including Mindreef SOAPscope Server™



By David S. Linthicum  
and Jim Murphy

**A step-by-step guide for defining your own testing domain,  
understanding your unique needs, and testing all components  
to help ensure your SOA will be productive from day one**

## Key Strategies for SOA Testing

Published by Mindreef® Inc.

22 Proctor Hill Road

Hollis, NH 03049

www.mindreef.com

Copyright © 2007 by Mindreef, Inc., Hollis, New Hampshire

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means electronic, mechanical, photocopying, recording, scanning or otherwise except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to Mindreef, Inc. Requests to the Publisher for permission should be addressed to the Legal Department, Mindreef, Inc., 22 Proctor Hill Road, Hollis, NH 03049, (603) 465-2204, fax (603) 465-6583, or via email to: info@mindreef.com.

Trademarks: Copyright 2007 Mindreef, Inc. All rights reserved. Mindreef and SOAPscope are registered trademarks; Mindreef SOAPscope Server, Mindreef SOAPscope Workstation, Mindreef Load Check, Mindreef Policy Rules Manager, Service Spaces, Pseudocode View, and The SOA Quality Company are trademarks of Mindreef, Inc. © Photographer: Vadim Rybakov | Agency: Dreamstime.com. All other trademarks are the property of their respective owners.

**LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OR FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND THE STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS DISTRIBUTED AND/OR SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHORS SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHORS OR PUBLISHERS ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.**

ISBN 13: 978-0-615-17593-5

Manufactured in the United States of America

Mindreef, Inc., The SOA Quality Company™ | 22 Proctor Hill Road, Hollis, NH 03049  
(603) 465-2204 | www.mindreef.com

# Key Strategies for **SOA Testing**

Including Mindreef SOAPscope Server™



By David S. Linthicum  
and Jim Murphy

**Mindreef**®  
The SOA Quality Company™

## About the Authors

**David S. Linthicum** is a Managing Partner at ZapThink, LLC. He joined the firm through the acquisition of Linthicum Group, LLC, a consulting organization dedicated to excellence in SOA product development, SOA implementation, corporate SOA strategy, and leveraging the next generation Web (Web 2.0).



Dave is the former CEO of BRIDGEWERX, former CTO of Mercator Software, and has held key technology management roles with a number of organizations including CTO of SAGA Software, Mobil Oil, EDS, AT&T, and Ernst and Young. Dave is on the board of directors serving Bondmart.com, and provides advisory services for several venture capital organizations and key technology companies. In addition, Dave was an associate professor of computer science for eight years, and continues to lecture at major technical colleges and universities including the University of Virginia, Arizona State University, and the University of Wisconsin. Dave keynotes at many leading technology conferences on application integration, SOA, Web 2.0, and enterprise architecture, and has appeared on a number of TV and radio shows as a computing expert.

Dave has authored over 800 articles and columns for major computing publications, and has monthly columns in several popular industry magazines including *Web Services/SOA Journal* and *Business Integration Journal*. Dave writes the Real World SOA blog on InfoWorld.com, and hosts the *InfoWorld* SOA Report Podcast. He is also a columnist and blogger for Intelligent Enterprise. Dave has authored or co-authored ten books including David Linthicum's Guide to Client/Server and Intranet Development, and the groundbreaking and best selling *Enterprise Application Integration*, released in 1998. His latest book, *Next Generation Application Integration*, is also a best seller.

**Jim Murphy** is Vice President of Product Management at Mindreef, with more than 10 years experience designing, implementing, testing and debugging distributed software architectures using Java, .NET, C++ and XML. Prior to Mindreef, he was an independent consultant, working with companies building distributed systems focused on DCOM and high-performance XML based messaging. He also served as a director, software architect and senior software engineer at several product and consulting organizations including Aspen Technology, Foliage Software Systems, TransactionWorks, and WWWhoosh, the Excelergy Corporation. Jim holds a BAsC in Engineering from the University of Waterloo.




Jim has authored articles for trade publications, including *Software Test & Performance Magazine*, and has spoken about Web services and SOA testing and quality at key industry events including Software Development Best Practices.

## *About Mindreef*

Mindreef, Inc., The SOA Quality Company™, is a leading provider of integrated software solutions for the successful development of Web services, enabling organizations to meet their goals for high-quality SOA adoption. The Mindreef award-winning family of SOAPscope® products enable project teams, architects, application developers, testers, managers, operations, and support staff to build, deploy, and maintain software for an SOA. More than 3,000 customers at more than 1,200 organizations worldwide, including 40 of the Fortune 100, use Mindreef products.

Mindreef SOAPscope products were developed through the understanding that SOA quality goes well beyond the quality of individual components such as Web services. Mindreef helps organizations ensure quality as their SOA teams define, implement, integrate, test and deploy Web services and composite applications.

# TABLE OF CONTENTS



<b>Introduction</b> .....	<b>1</b>
<b>Chapter 1: SOA Testing, The Basics</b> .....	<b>2</b>
Service Distribution .....	2
Service Data .....	3
Service Behavior .....	3
Service Integrity .....	3
Service Design .....	4
SLAs and Performance Testing .....	4
Adherence to a Holistic Architecture .....	4
<b>Chapter 2: Creating a SOA Test Plan</b> .....	<b>5</b>
Understanding Your Own Needs .....	5
Transactional Heavy .....	6
Data Heavy .....	6
Process Heavy .....	6
Defining the Approach .....	6
Bottom Up .....	6
Top Down .....	7
System .....	7
Understanding the Domain .....	7
Considering Service Performance .....	8
Testing Levels .....	8
Information Level .....	8
Service Level .....	9
Process Level .....	10
Bringing the Plan Together .....	11
<b>Chapter 3: Step-by-Step Guide to SOA Testing</b> .....	<b>15</b>
So How do you go About Testing a SOA? .....	15
1. Define the Testing Domain.....	16
System Description Analysis .....	16
SOA Testing Proof of Concept .....	16

2. Define Architectural Objectives .....	17
Define Logical Architecture .....	17
Define Physical Architecture .....	17
3. Design, Review and Test Planning .....	18
Review Core Design .....	18
Create Test Plan .....	18
Create Test Plan and Implementation Schedule .	18
4. Create Functional Testing Approach .....	19
Define Core Services .....	19
Define Core Data .....	19
Define Core Processes .....	20
5. Define Performance Requirements .....	20
Create Service Level Performance Requirements	20
Create Data Level Performance Requirements ...	21
Create Process Level Performance Requirements	21
6. Define SLA Requirements .....	21
Create a Service Level SLA .....	21
Create a Service Level SLA .....	22
Create a Service Level SLA .....	22
7 .Define Data Layer Testing Approach .....	22
Create Data Layer Definition .....	23
Create Data Abstraction Testing Approach .....	23
Create Data Access Testing Approach .....	23
8. Define Services Layer Testing Approach .....	23
Create Services Layer Definition .....	24
Create Services Testing Approach .....	24
Create Composite Services Testing Approach .....	24
9. Define Policy Layer Testing Approach .....	25
Create Policy Layer Definition .....	25
Create Policy Testing Approach .....	26
Create Governance Testing Approach .....	26
10. Define Process Layer Testing Approach .....	26
Create Process Layer Definition .....	27
Create Process Testing Approach .....	27

11. Define Service Simulation .....	28
Create Service Simulation Approach .....	28
Create Service Simulation Model .....	28
Test Service Simulation .....	28
12. Create Core Scenarios .....	29
Create Approach to Scenarios .....	29
Create Specific Scenarios .....	29
13. Create User-Defined Compliance Rules .....	30
Create Approach to Compliance Rules .....	30
Create Compliance Rules .....	30
14. Select the SOA Testing Technology Suite .....	31
Define Requirements .....	31
Define Candidate Technology .....	31
Technology Analysis .....	31
Technology Selection .....	31
Technology Validation .....	31
15. Testing Execution .....	32
Unit Testing .....	32
Functional Testing .....	33
Regression Testing .....	33
Compliance and Validation .....	33
16. Looping Back to Design and Development .....	34
Analysis of Test Results .....	34
Define Impact on Design and Development .....	34
17. Define Diagnostics for Design Time and Run Time .....	35
18. SOA Testing Debrief and Lessons Learned .....	36
19. Operational Test Planning .....	37
Create Approach to Operational Test Planning ...	37
Development of Operational Test Plan .....	37
Select Operational Testing Tools .....	37
<b>Chapter 4: Using Mindreef SOAPscope Server .....</b>	<b>38</b>
<b>Conclusion .....</b>	<b>40</b>





## INTRODUCTION



Service Oriented Architecture (SOA) requires a unique approach to testing. Unless you're willing to reorient your testing procedures and technology now, you may find yourself behind the curve as SOA becomes systemic to all that is enterprise architecture.

Moreover, as we add more complexity to get to an agile and reusable state, the core notion of SOA becomes that much more strategic to the business. If you're willing to take the risk, the return on your SOA investment will come back three fold... that is, if it is a well-tested SOA. Untested SOA could cost you millions.

In this handbook, we will provide you with the key notions around SOA testing as well as the processes needed to effectively test a SOA, including a step-by-step guide for defining your own testing domain, understanding your unique needs, and then testing all components in such a way as to ensure that your SOA will be productive and useful from the first day of operation.

We'll also introduce technology such as Mindreef SOAPscope Server, that when used with the SOA testing approaches defined in this paper, will provide a strategic advantage through automation and repeatability.

## SOA TESTING, THE BASICS



While there are no hard and fast guidelines as to what makes up a well-defined and developed service, we do know a few things:

- Services are not complete applications or systems. They are small parts of an application, and should be tested as such. Services are not subsystems; they are small parts of subsystems as well. Indeed, services are more analogous to traditional application functions in terms of design, and how they are leveraged to form solutions, fine- or coarse-grained. Knowing that, we have a better basis of understanding when approaching the services testing problem.
- Each service has a specific purpose, and they are not complex or naturally dependent upon other services. Thus, they are easily abstracted into composite applications, in essence, leveraging these services as if they are functions local to the composite.
- Services should exist with a high degree of autonomy. They should execute without dependencies, if at all possible. This allows you to leverage the service by itself, and design the service with this in mind, no matter how fine- or coarse-grained the service is.

### *Service Distribution*

When considering SOA testing, you need to consider that the services are distributed within the enterprise. This service distribution comes with its own set of challenges, including the ability discover the services under test in heterogeneous environments. Moreover, the actual runtime testing of each service is complex unto itself.

## *Service Data*

Testing of **the data** as it flows through a service is critical to the health of the service. While many attempt to leverage traditional data testing methods, testing data in the context of services is a bit different, such as testing database validity as a service. Approaches to service data testing includes monitoring points of information externalization which are leveraged to watch the data as it flows through the service. Also, consider the information the service consumes, with the information the service produces.

## *Service Behavior*

When considering testing, we also need to focus on **the behavior**, or the functionality of the service. This means that the logic of the service is monitored and determined to be sound, and that the information flowing into the service is processed correctly according to the design of the service. We can accomplish this by building up permutations of request input data AND XML structure, mixing with different security context. Establishing coverage metrics is key here. Moreover, if the service is designed to provide different behavior via context, that needs to be monitored throughout the processing of the service as well. Same approach; set up monitoring points, and watch the behavior through execution.

## *Service Integrity*

**Service integrity** is the degree to which the service is able to deliver consistent value over a long period of time. Typically this is tested through a sustainable testing cycle, reflecting real world use cases. In other words, the ability for the service to deliver functionality to a consumer. Thus, testing for service integrity is critical to the overall testing of the SOA.

## *Service Design*

Testing for **service design** means understanding the functionality of the service, and the design patterns used, and the ability for that service to live up to that particular design. This process involves interaction between multiple roles, such as architects, designers, and developers.

Moreover, this incorporates aspects of governance relating to how service interfaces conform to local and industry design standards. The idea of service design testing is to understand the overall design of the service, and thus how the services exist in support of those patterns.

## *SLAs and Performance Testing*

**Service level agreements (SLAs)** are contracts that exist internally or externally and consider the consistent performance of a service. The idea is to create an agreement that insures that a service will provide a specific level of performance, and then measure the performance of the service to the agreement.

This is an important component of testing since services that don't live up to SLAs will hinder the overall performance and functionality of the SOA. Keep in mind that SLAs often control your level of quality.

## *Adherence to a Holistic Architecture*

Finally, it's important to note that a SOA is an architecture, thus you need to test the architecture holistically, including how the overall architecture is living up to core objectives such as reuse and agility. This means working up from the most primitive components of the architecture, such as information persistence, up through data and transactional services, up to orchestration and composite application development.

## CREATING A SOA TEST PLAN



Creating a test plan around SOA is more involved than one might think. Needed is a way to understand the architecture at a meta level, and then figure out how to break the architecture into component parts that can be tested as a unit, tested as a linked-group, as well as tested as the architecture holistically.

One of the things to consider is the notion of enterprise and project-wide test planning. Typically this means creating an enterprise-wide set of testing standards and procedures that drive value down to testing at the project levels. You must consider the core value of each, and the ability for the projects to drive their issues up to the enterprise-wide testing strategy, and the ability for the enterprise-wide testing strategy to drive value down to the projects.

### *Understanding your own Needs*

While many in the press and vendor community would lead you to believe that SOAs are all the same, the reality is that the architecture and solution sets are very different from problem domain to problem domain.

So part of the process of creating your own SOA test plan is to understand that, and understand the requirements that are unique to you, and thus must be part of the test planning.

Typically, SOAs have design patterns that fall into a few major categories:

- Transactional Heavy
- Data Heavy
- Process Heavy

**Transactional Heavy** SOAs are architectures where the use of transactional services is more apparent. Typically, these are on-line transaction processing types of application clusters that use an architecture where transactional services are leveraged and invoked more than others.

**Data Heavy** SOAs are architectures where most of the services employed are data services, or services that broker in information more so than behavior.

**Process Heavy** SOAs are architectures where the core dynamics of the architecture are driven at the process level. Typically, these are architectures where volatility is the norm, and thus the core services are abstracted into a process layer where they can be changed more easily.

## Defining the Approach

Throughout the methodology we call out steps to define each component's approach to testing. Prior to that, and within the plan, we need to focus on how we're going to approach testing holistically.

Considering the approach to testing SOA, we have a few options, including:

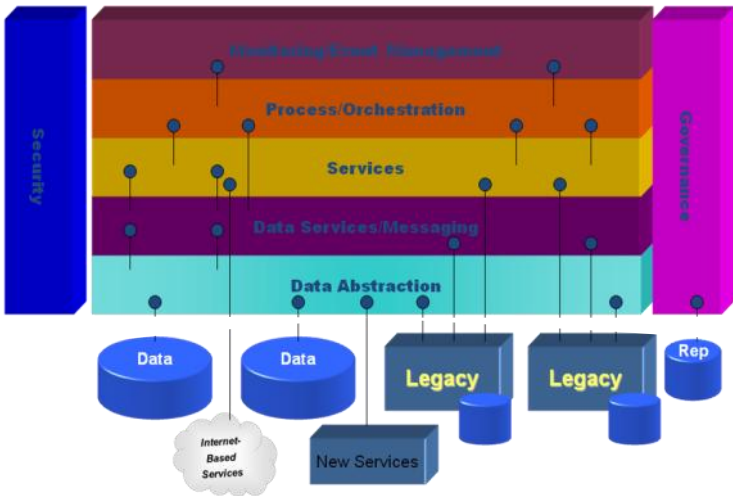
- Bottom Up
- Top Down
- System

The **Bottom Up** approach to testing means we're looking to test our SOA from the most primitive features to the most sophisticated. Typically this refers to working up from the data to the process (see Figure 1), working up through the data services, data abstraction, and transactional services layers to the process layers, and finally the monitoring and event management layers. This is done by testing each component at the lower layers, and moving up to the high layers.

The **Top Down** approach reverses the order, working from the monitoring and event management layers down through the architecture to the data. This is done by testing each component at the higher layers, moving down to the lower layers.

The **System** approach means we're testing the architecture holistically, looking at the entire solution set as one functional [what], and testing all interfaces to the architecture at all levels noting the inputs, outputs, and behaviors during testing.

It should be noted that you can use any or all approaches; they are not mutually exclusive. Indeed, most successful SOA testing projects leverage all approaches, putting emphasis on one approach based on the requirements of the architecture.



**Figure 1:** To approach SOA testing you can work up from the bottom, down from the top, test components, or the entire architecture.

## *Understanding the Domain*

When creating a test plan, the primary point of reference should be a complete understanding of the problem domain. While it would be nice to use the same approaches for all SOAs, the reality is that you must first analyze the problem domain in detail before you can figure out how to test it.



Typically, you need to have semantic level, services level, and process level understanding of the problem domain before you're able to effectively select the proper testing approach and test planning. This comes from the analysis done when the architecture was created, but if not completed prior to the testing phase, it should be understood nonetheless.

## *Considering Service Performance*

Since services are the central component of a SOA, the ability for the services to provide the necessary performance is key to the success of a SOA. Thus, during the test planning, performance testing should be:

- Defined at a high level, or the approach
- Defined at a low level, [or?] the method

By doing this we can insure that we're validating that the services will live up to the SLAs defined, and also provide holistic performance of the SOA that meets performance expectations at all levels of the architecture. Keep in mind the overall performance of the SOA is defined by the slowest service in the architecture.

## *Testing Levels*

As we discussed above, you need to consider the levels of a SOA to better define testing approaches. For our purposes we can call these levels:

1. Information Level
2. Service Level
3. Process Level

## *Information Level*

The information level of a SOA refers to the underlying data that links to the services. While in many cases these are standards databases, there are legacy systems, CRM systems, ERP systems, and other "stovepipe" applications which may produce and consume business information as well.

The information level is core to SOA since most services that exist within a SOA are data services, and most business systems are all about the processing of information. Thus, to successfully test a SOA at the information level, you need to understand that it's not only the value of the information, as persisted, but it's the use of the data within the services.

You test data within the services considering the patterns of information processing through the services. Some key questions you need to ask include:

- What metadata is bound to the service?
- What integrity constraints are enforced by the service?
- How is the data abstracted from the source database?

## *Service Level*

Within the world of SOA, services are the building blocks, and are found at the lowest level of the stack. Services become the base of a SOA, and while some are abstract existing "legacy services," others are new and built for specific purposes. Moving up the stack, we then find composite services, or services made up of other services, and all services abstract up into the business process or orchestration layer, which provides the agile nature of a SOA since you can create and change solutions using a configuration metaphor.

When testing services, you need to keep the following in mind:

***Services are not complete applications or systems, and must be tested as such.*** They are a small part of an application. Nor are they subsystems; they are small parts of subsystems as well. Thus, you need to test them with a high degree of independence, meaning that the services are both able to properly function by themselves, and also as part of a cohesive system. Indeed, services are more analogous to traditional application functions in terms of design and how they are leveraged to form solutions, fine- or coarse-grained.

***The best approach to testing services is to list the use cases for those services.*** At that point you can design testing approaches for that service including testing harnesses, or the use of SOA testing tools (discussed later). You also need to consider any services that the service may employ, and thus be tested holistically as a single logical service. In some cases you may be testing a service that calls a service, that calls a service, where some of the services are developed and managed in house, and some of them exist on remote systems that you don't control. All use cases and configurations must be considered.

***Services should be tested with a high degree of autonomy.***

They should execute without dependencies, if at all possible, and be tested as independent units of code using a single design pattern that fits within other systems which use many design patterns. While all services can't be all things to all containers, it's important to spend time understanding their foreseeable use, and make sure those are built into the test cases. You should have the ability to simulate services. Testers can build simulations of dependent services to isolate a service under test. The tester uses SOAP messages and the WSDL to "mock" the services at a live HTTP endpoint.

***Services should have the appropriate granularity.*** Don't focus on too-fine-grained or too-course-grained. Focus on that correct granularity for the purpose and use within the SOA. Here the issues related to testing are more along the lines of performance than anything else. Too-fine-grained services have a tendency to bog down performance due to the communications overhead required when dealing with so many services. Too-loose-grained, they don't provide the proper autonomic values to support their reuse. You need to work with the service designer on this issue.

## *Process Level*

One can consider the process level of a SOA as the place where services are abstracted, orchestrated, or bound together, to form a business solution.

In essence, it's the process layer where solutions are formed, changed, changed again, removed, and added.

We leverage a process level within SOA because it places volatility into a single domain, and thus allows the architect to adjust core business processes to meet the changing needs of the business. This also becomes a single point of failure, and the process level must be tested with the same degree of importance as the service and information levels.

For the purposes of testing, we can consider the process level as another collection of services that have to consume information, process information, provide functional behavior, and produce information. Moreover, the process level typically binds services together into composite services, for the purpose of processes, and also drives sequencing, nesting, and management of service interdependence.

While it would be nice if there were a single approach to creating and maintaining a process level, the truth is that process levels are created using all sorts of approaches, standards, and enabling technologies for SOA. BPEL for use with services orchestration is an example of an approach, choreography is another, as well as proprietary process engines. By respecting the service interface boundary, the service implementation is largely irrelevant. The SOA tenets create constrained and verifiable layers in the architecture.

### *Bringing the Plan Together*

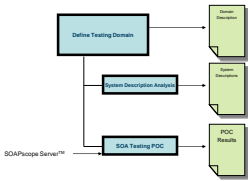
Creating the core plan for SOA testing means considering all of the issues we just defined above. However, the exact way in which you test your SOA, and the plan supporting it, will vary from problem domain to problem domain. Thus, the most important step in creating a test plan for your SOA is to understand what's unique about your situation, how your architecture exists at the information, service, and process levels, and how to test each and all levels effectively.

A common pattern for all SOA is the fact that SOA testing boils down to services. Thus, testing services as collections, stand-alone, or holistically as a complete architecture is core to all of this. Equally important is how you create a plan, and the testing technology you employ to test services in terms of reliability, data validation, and ability to live up to SLAs.

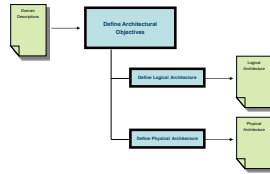
Looking at this problem holistically, you have a set of predefined steps that can be taken in sequence or can be interactive.

At a high level, the steps look like this:

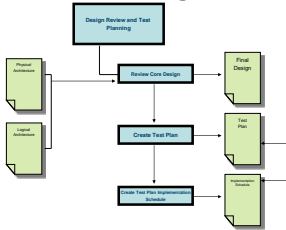
### 1. Define the Testing Domain



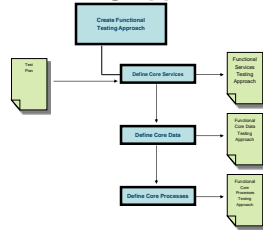
### 2. Define Architectural Objectives



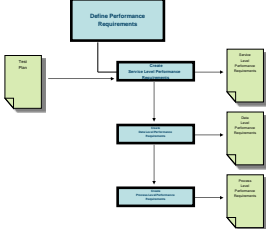
### 3. Design Review and Test Planning



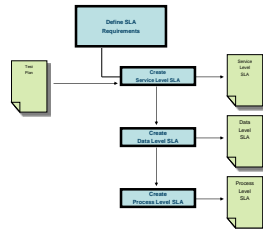
### 4. Create Functional Testing Approach



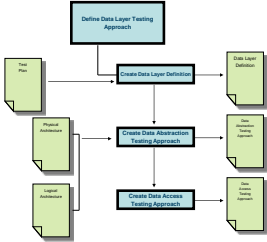
### 5. Define Performance Requirements



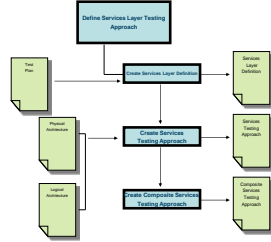
### 6. Define SLA Requirements



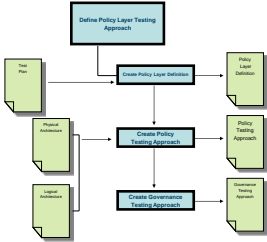
### 7. Define Data Layer Testing Approach



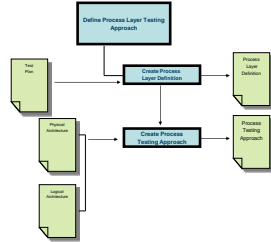
### 8. Define Services Layer Testing Approach



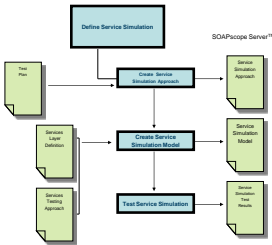
### 9. Define Policy Layer Testing Approach



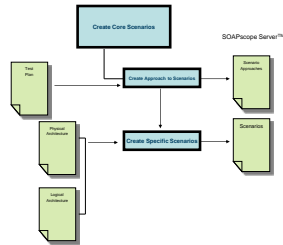
### 10. Define Process Layer Testing Approach



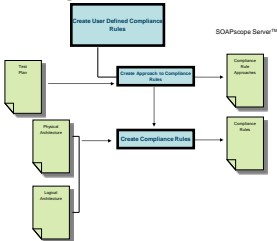
### 11. Define Service Simulation



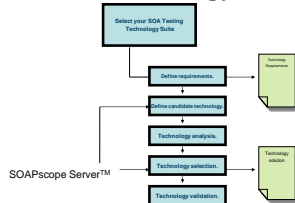
### 12. Create Core Scenarios



### 13. Creating User Defined Compliance Rules



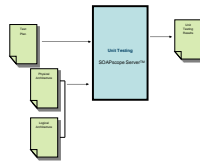
### 14. Select SOA Testing Technology Suite



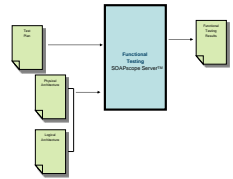
### 15. Testing Execution

- **Unit Testing**
- **Functional Testing**
- **Regression Testing**
- **Compliance and validation (WSDL, schema, messages)**

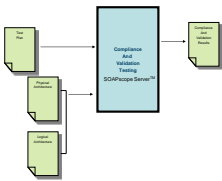
Unit Testing



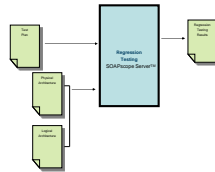
Functional Testing



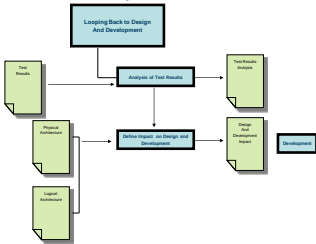
Compliance and validation (WSDL, schema, messages)



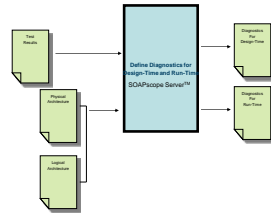
Regression Testing



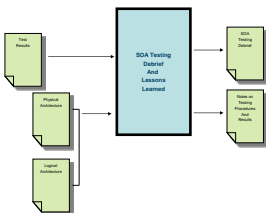
### 16. Looping back to Design and Development



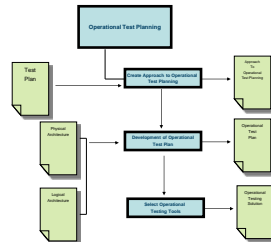
### 17. Define Diagnostics for design-time and run-time



### 18. SOA Testing Debrief and Lessons Learned



### 19. Operational Test Planning



Next, let's see how you do this step-by-step.

## STEP-BY-STEP GUIDE TO SOA TESTING



### *So, how do you go about testing a SOA?*

Beyond the approaches, concepts, and guidelines provided above, it's productive to provide you with a step-by-step methodology. In essence, if you follow these steps you can insure that your SOA will be properly tested, you will create all of the artifacts you need, understand your own problem domains, and leverage key testing technology.

As you leverage these steps, keep a few things in mind:

- This methodology is iterative, meaning, you don't have to move through the steps sequentially; however, a few steps are dependent upon artifacts created in the prior steps, but most are independent and loosely coupled.
- While you can skip a few steps (as needed for your own requirements), most of the steps address a critical component of SOA testing, so make sure you understand at least what the step is suggesting before moving on.
- Pay close attention to why things are done, more so than how they are being done.
- There is a suggested "loop" in this method. This means we assume that once we complete the steps, we loop back to complete the steps again for each instance of a SOA. As such, we will incorporate lessons learned as we drive again through the method.
- Take the time to make this method your own. Meaning, customize it for your purposes, as needed, even add or change steps.



## 1. Define the Testing Domain

The first step is to define the testing domain, or the area of the SOA that will be under test. There are a few sub-steps, including:

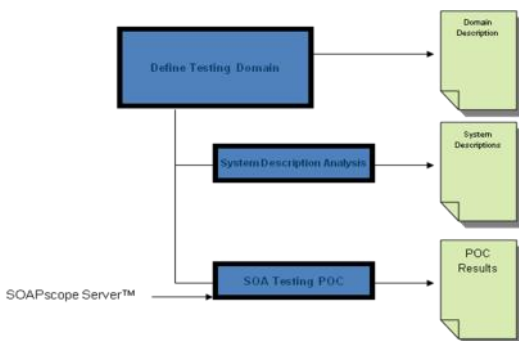
- System Description Analysis
- SOA Testing Proof of Concept (POC)

Key artifacts created are:

- Domain Description
- System Description
- Proof-of-Concept Results

**System description analysis** means the complete analysis of all systems under test and the creation of the system description document that defines the overall description of the system, including components, data, services, and processes. This is a not a detailed description, but an overall view of the system which will allow enough understanding to approaching testing.

**SOA testing proof of concept** is the process of leveraging a testing tool within the POC. This is where the Mindreef SOAPscope® family of products first come into play as you determine how the tools are leveraged within the context of the SOA. The data points leveraged out of this POC will provide critical information in terms of how the SOA testing is defined during this process.



**Figure 2:** Define the Testing Domain

## 2. Define Architectural Objectives

The second step is to define the architectural objectives, or the logical view of the architecture and the physical view of the architecture. The major sub-steps include:

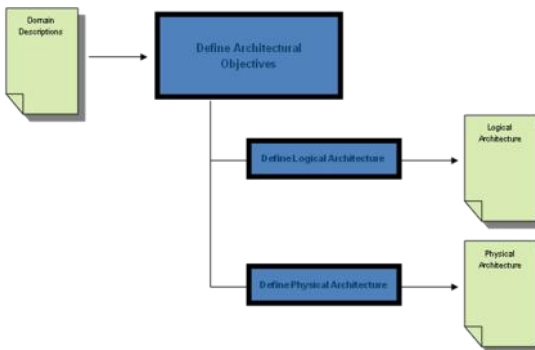
- Define Logical Architecture
- Define Physical Architecture

Key artifacts created are:

- Logical Architecture
- Physical Architecture

**Define logical architecture** refers to the process of defining how the architecture exists conceptually, independent of physical instances of technology. We use this document as a way to understand the logical components of the SOA, and how each component interacts with others.

**Define physical architecture** refers to the process of defining how the physical instances of technology (ESBs, orchestration, service platform) interact, one to another. Like the logical architecture, this document demonstrates how all of the components interact, but now defines all components, interfaces, standards, and all enabling technology. Both the logical and physical architectures will be key to the remainder of the process.



**Figure 3: Define Architectural Objectives**

### 3. *Design Review and Test Planning*

In this step we set up the testing project, including reviewing the core design of the SOA, both logical and physical architectures, and determine how we are going to approach testing, the test plan to leverage, and the project work schedule. The major sub-steps include:

- Review Core Design
- Create Test Plan
- Create Test Plan Implementation Schedule

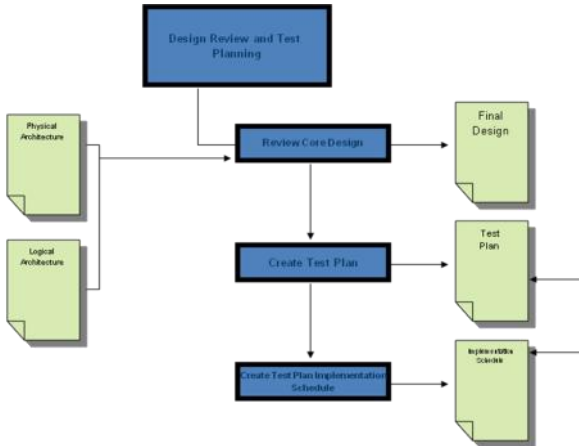
Key artifacts created are:

- Final Design
- Test Plan
- Implementation Schedule

***Review core design***, as we discussed above, means taking time to review both the logical and physical architectures and consider the design in how we define our testing procedures. The criteria should be created during the plan before reviewing the core design.

***Create test plan*** is the process of leveraging the design and architecture information to put together an approach and plan for testing the SOA, including all processes, procedures, and enabling technology. This is a foundation document for the rest of the methodology.

***Create test plan and implementation schedule*** is the process of creating a project and work plan for how the resources are to be leveraged to complete testing of the SOA.



**Figure 4:** Design Review and Test Planning

#### 4. Create Functional Testing Approach

In this step we set up the approach to functional testing of the SOA. The major sub-steps include:

- Define Core Services
- Define Core Data
- Define Core Processes

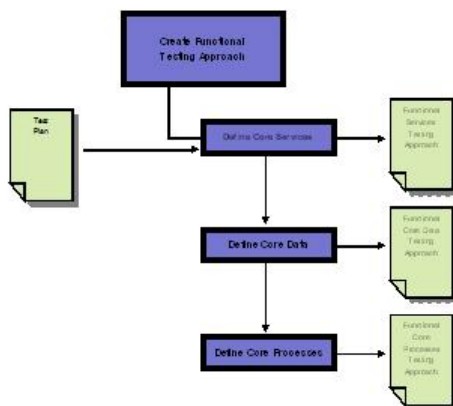
Key artifacts created are:

- Functional Core Services Testing Approach
- Functional Core Data Testing Approach
- Functional Core Processes Testing Approach

**Define core services** is the process of listing and defining all of the services within the problem domain that will be under test, and the approach for functional testing those services. This means understanding what they do, who they are for, what's the native metadata, etc.

**Define core data** is the process of listing and defining all of the data within the problem domain that will be under test, and the approach for functional testing that data.

**Define core processes** is the process of listing and defining all of the processes within the problem domain that will be under test, and the approach for functional testing those processes. When considering this process, we mean how the service is composed with other services to form a higher level process.



**Figure 5: Create Functional Approach**

## 5. Define Performance Requirements

In this step we approach performance requirements. The major sub-steps include:

- Create Service Level Performance Requirements
- Create Data Level Performance Requirements
- Create Process Level Performance Requirements

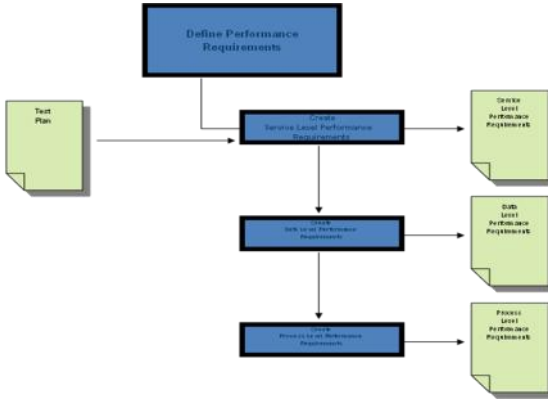
Key artifacts created are:

- Service Level Performance Requirements
- Data Level Performance Requirements
- Process Level Performance Requirements

**Create service level performance requirements** refers to the process of determining performance expectations for all services within the testing domain, including response time under an increasing load.

**Create data level performance requirements** refers to the process of determining performance expectations for all data access services, including response time under an increasing load.

**Create process level performance requirements** refers to the process of determining performance expectations for all processes, including response time under an increasing load.



**Figure 6: Define Performance Requirements**

## 6. Define SLA Requirements

In this step we approach SLA (service level agreement) requirements. The major sub-steps include:

- Create Service Level SLA
- Create Data Level SLA
- Create Process Level SLA

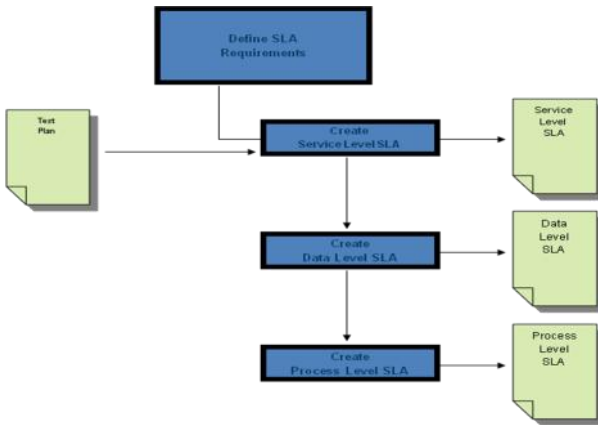
Key artifacts created are:

- Service Level SLA
- Data Level SLA
- Process Level SLA

**Create service level SLA** refers to the process of defining the performance expectations and agreement for particular services, including response times and uptime requirements.

**Create data level SLA** refers to the process of defining the performance expectations and agreement for particular data (persistence), including response times, and uptime requirements. This means that certain data is more important than other data and should be tested differently.

**Create process level SLA** refers to the process of defining the performance expectations and agreement for particular processes, including response times and uptime requirements.



**Figure 7: Define SLA Requirements**

## 7. Define Data Layer Testing Approach

In this step we define data layer testing requirements and the testing approach. The major sub-steps include:

- Create Data Layer Definition
- Create Data Abstraction Testing Approach
- Create Data Access Testing Approach

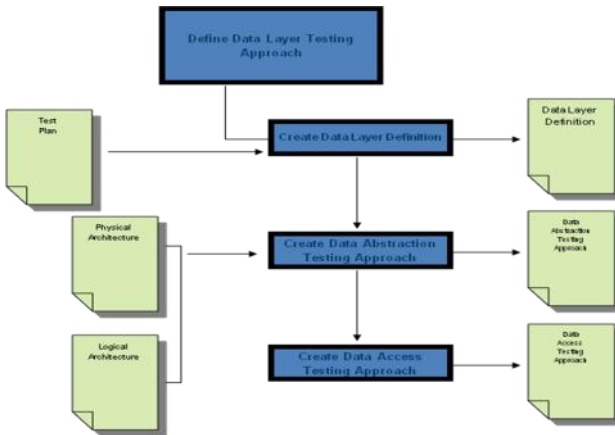
Key artifacts created are:

- Data Layer Definition
- Data Abstraction Testing Approach
- Data Access Testing Approach

**Create data layer definition** refers to the process of defining, in detail, the data layer that will be under test. This includes structure, attributes, validation logic, security, and other information that will assist in defining information to be tested.

**Create data abstraction testing approach** refers to the process of defining any data abstractions in place, and the approach for testing the data abstraction, that have remapped the physical database to different virtual structures.

**Create data access testing approach** refers to the process of creating the approach to data level testing, including tools, standards, technology, and other information that will assist in defining the approach to data access testing.



**Figure 8: Define Data Layer Testing Approach**

## 8. Define Services Layer Testing Approach

In this step we define the services layer testing requirements and approach. The major sub-steps include:

- Create Services Layer Definition
- Create Services Testing Approach
- Create Composite Services Testing Approach



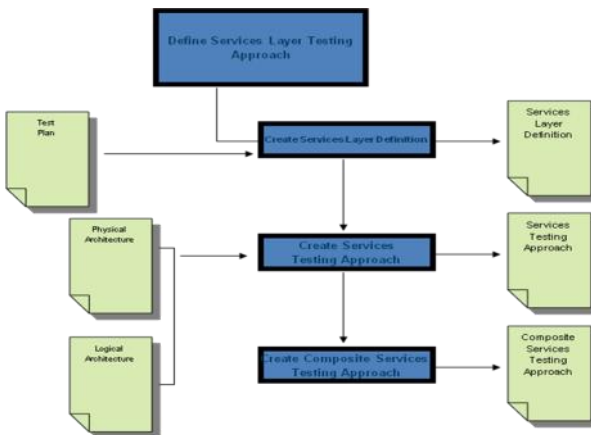
Key artifacts created are:

- Services Layer Definition
- Services Testing Approach
- Composite Services Testing Approach

**Create services layer definition** refers to the process of defining all services that are under test, including access approaches, enabling technology, and use of standards.

**Create services testing approach** refers to the process of determining the best way to approach testing the services defined in the previous step. This means understanding access mechanisms, service grouping, and other information that will be useful in creating the approach.

**Create composite services testing approach**, like the previous step, refers to the process of determining the best way to approach testing of the composite services. This means understanding access mechanisms, service grouping, and other information that will be useful in creating the approach.



**Figure 9:** Define Services Layer Testing Approach

## 9. Define Policy Layer Testing Approach

In this step we define the policy layer testing approach. Most companies that are building Web services encounter inoperability issues when services are implemented and consumed using tools from different vendors. To mitigate that, organizations mandate that WSDL contracts conform to the WS-I Basic Profile.

It is common practice to augment industry standard policies with additional requirements or best practices created by corporate and lead architects who focus on infrastructure. Their goal is to ensure that contracts interoperate with the specific toolkits and frameworks that a company has adopted.

Rigorous testing alone cannot impose quality where it doesn't exist. Even well-written services cannot guarantee broad interoperability unless standards and best practices are well designed and adhered to throughout an organization and throughout the development lifecycle. The major sub-steps include:

- Create Policy Layer Definition
- Create Policy Testing Approach
- Create Governance Testing Approach

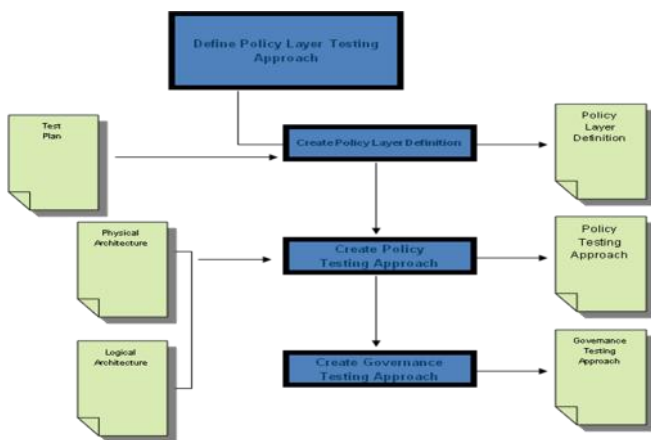
Key artifacts created are:

- Policy Layer Definition
- Policy Testing Approach
- Governance Testing Approach

**Create policy layer definition** refers to the process of defining all policies that are under test, including access approaches, enabling technology, and use of standards. Policies need to be tested since they are enforced against the use of services. Thus, once policies are created for a service; those policies need to be tested to determine that they are functioning per the design.

**Create policy testing approach** refers to the process of determining the best way to approach testing the policies defined in the previous step. This means understanding access mechanisms, policy grouping, and other information that will be useful in creating the approach. You test policies by running specific testing scenarios against the service to determine if the policy is behaving correctly.

**Create governance testing approach** refers to the process of determining the best way to approach testing of the holistic governance layer.



**Figure 10: Define Policy Layer Testing Approach**

## 10. Define Process Layer Testing Approach

In this step we define the process layer testing requirements and approach. The major sub-steps include:

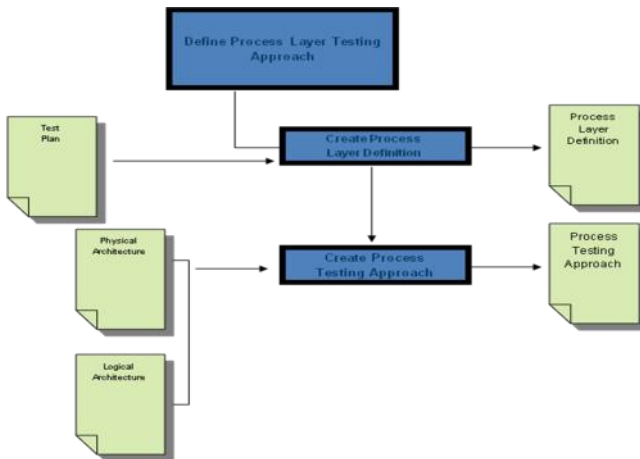
- Create Process Layer Definition
- Create Process Testing Approach

Key artifacts created are:

- Process Layer Definition
- Process Testing Approach

**Create process layer definition** refers to the process of defining (what they are, who owns them, and how they are designed) all processes that are under test, including access approaches, enabling technology, and use of standards.

**Create process testing approach** refers to the process of determining the best way to approach testing the process defined in a previous step. This means understanding access mechanisms, process grouping, and other information that will be useful in creating the approach.



**Figure 11:** Define Process Layer Testing

## 1 1. Define Service Simulation

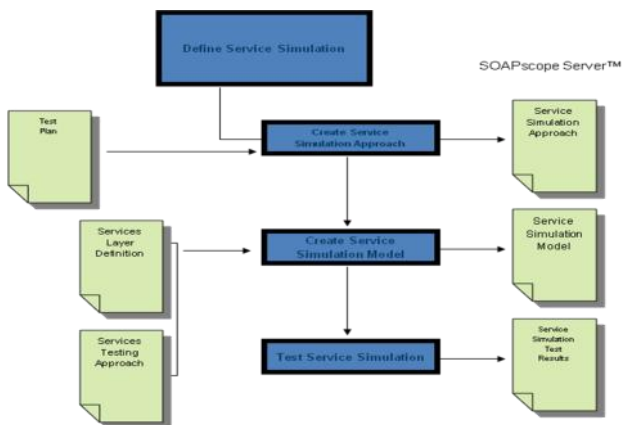
In this step we define the service simulation approach for testing. We do this to isolate service dependencies so you can test some services, without testing many services. The major sub-steps include:

- Create Service Simulation Approach
- Create Service Simulation Model
- Test Service Simulation

**Create service simulation approach** is the process of defining the core approaches to simulating services for use when testing the SOA. These simulation services provide a mechanism to test the services, and SOA, prior to deployment and ongoing.

**Create service simulation model** is the process of taking the approach created in the previous step, and defining the model for use while testing, and ongoing.

**Test service simulation** means using the service simulation approach and service simulation model to test the simulation services. Any issues are noted here, and addressed in this step.



**Figure 12: Define Service Simulation**

## 12. Create Core Scenarios

In this step we define the core scenarios for SOA testing. The major sub-steps include:

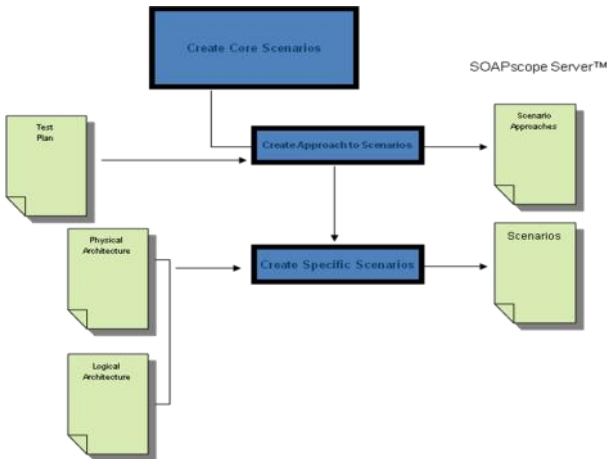
- Create Approach to Scenarios
- Create Specific Scenarios

Key artifacts created are:

- Scenario Approaches
- Scenarios

**Create approach to scenarios** refers to the process of creating a general approach to the scenarios or use cases employed to test the SOA. These scenarios approaches are designed to reflect real life uses of the services, and the SOA.

**Create specific scenarios** is just a process of leveraging the approach, defined in the previous steps, to create specific scenarios for testing the services and SOA.



**Figure 13:** Create Core Scenarios

### 13. Creating User-Defined Compliance Rules

In this step we define the user-defined compliance rules for SOA testing. This is part of the testing policy, and leveraged when design guidelines are part of a policy. The major sub-steps include:

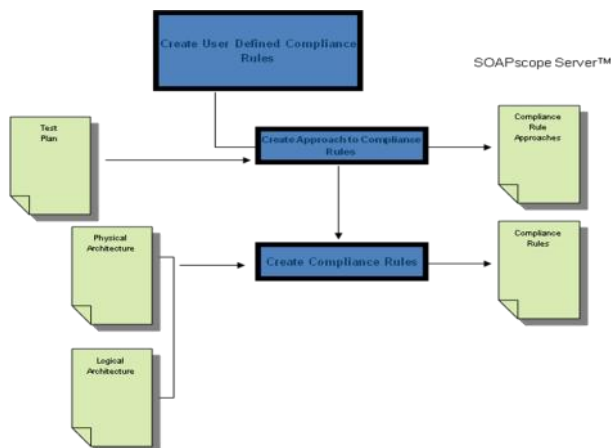
- Create Approach to Compliance Rules
- Create Compliance Rules

Key artifacts created are:

- Compliance Rules Approaches
- Compliance Rules

**Create approach to compliance rules** means creating a general way in which compliance rules will be approached in the context of SOA testing. These approaches should set the stage for creating the actual compliance rules that are created in the next step.

**Create compliance rules** means creating the actual compliance rules for the SOA, leveraging the approach created in the previous step.



**Figure 14: Create User-Defined Compliance Rules**

## 14. *Select the SOA Testing Technology Suite*

In this step we define the SOA Testing Technology Suite.

The major sub-steps include:

- Define Requirements
- Define Candidate Technology
- Technology Analysis
- Technology Selection
- Technology Validation

Key artifacts created are:

- Technology Requirements
- Technology Solution

***Define requirements***, meaning to define the requirements for the technology needed to be employed, including service testing suite, other technology required for SOA testing.

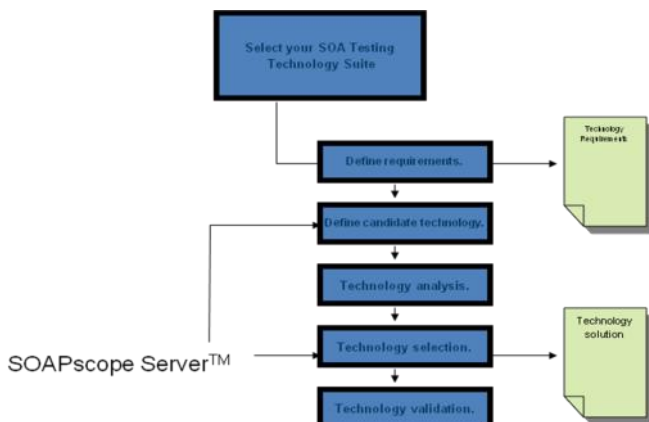
***Define candidate technology***, meaning to list the technology required for SOA testing, including integrated software solutions such as Mindreef SOAPscope Server™, Mindreef Load Check™, and Mindreef Policy Rules Manager™.

***Technology analysis***, meaning to analyze the technology for fit and function, for the application to the SOA testing procedures defined in the previous step.

***Technology selection***, meaning to select the Mindreef testing technology to test the SOA, with the artifact previously defined.

***Technology validation***, meaning to validate the technology to determine if it works properly in context to the requirements defined in the previous step. Typically, this is done through validation testing.



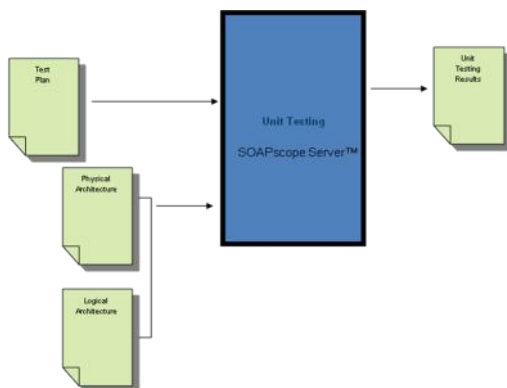


**Figure 15:** Select your SOA Testing Technology Suite

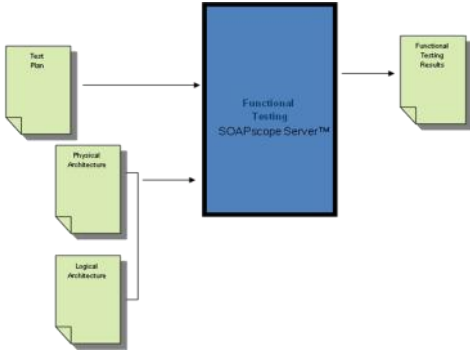
## 15. Testing Execution

In this step we carry out the testing of the SOA, including:

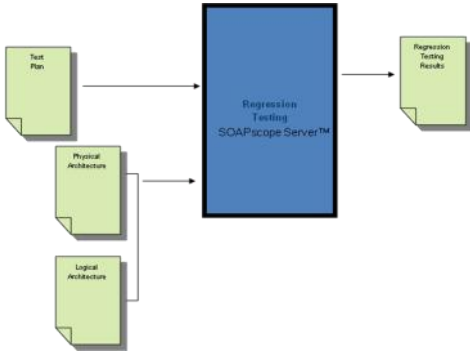
- a. Unit Testing
- b. Functional Testing
- c. Regression Testing
- d. Compliance and Validation (WSDL, schema, messages)



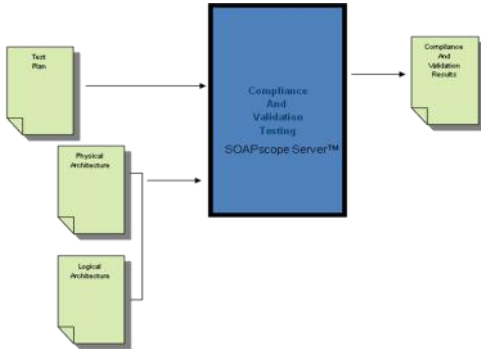
**Figure 16:** Unit Testing



**Figure 17: Functional Testing**



**Figure 18: Regression Testing**



**Figure 19: Compliance and Validation Testing**

## 16. Looping back to Design & Development

In this step we define how to feed the test results back to development to improve the SOA/services development process. The major sub-steps include:

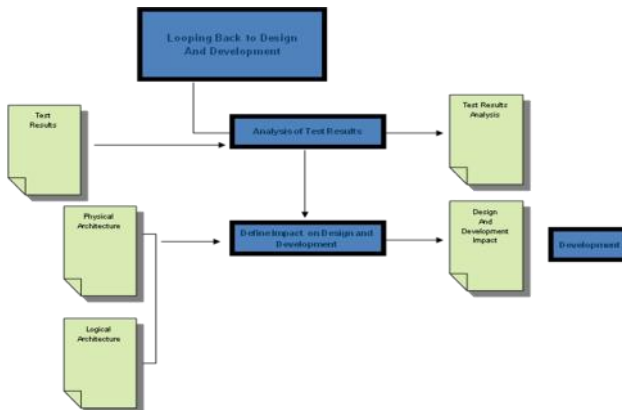
- Analysis of Test Results
- Define Impact on Design and Development

Key artifacts created are:

- Test Results Analysis
- Impact on Design and Development

**Analysis of test results**, meaning to look at the test results of the services/SOA testing process defined in this methodology, and prepare the results for feedback to design and development.

**Define impact on design and development**, meaning to send the results back from the services/SOA testing process to development.



**Figure 20:** Looping Back to Design and Development.

## 17. Define Diagnostics for Design-Time and Run-Time

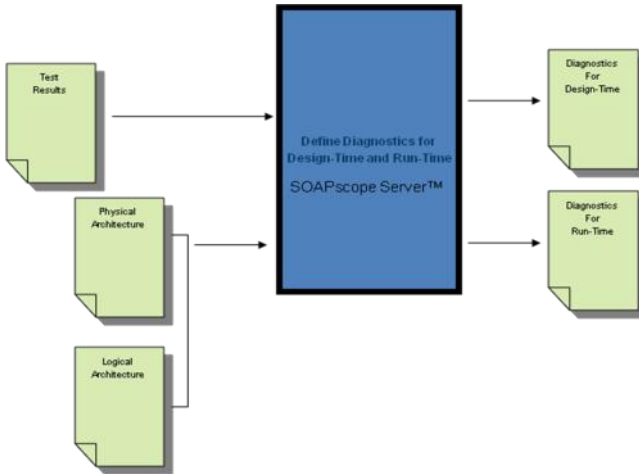
In this step we define core diagnostics for the SOA, and detailed approaches for implementation. The major sub-step is:

- Define Diagnostics for Design-Time and Run-Time

Key artifacts created are:

- Diagnostics for Run-Time
- Impact on Design and Development

**Define diagnostics for design-time and run-time** means to create the diagnostic approaches for the design-time and run-time instances of the SOA.



**Figure 21:** Define Diagnostics for Design-Time and Run Time

## 18. SOA Testing Debrief and Lessons Learned

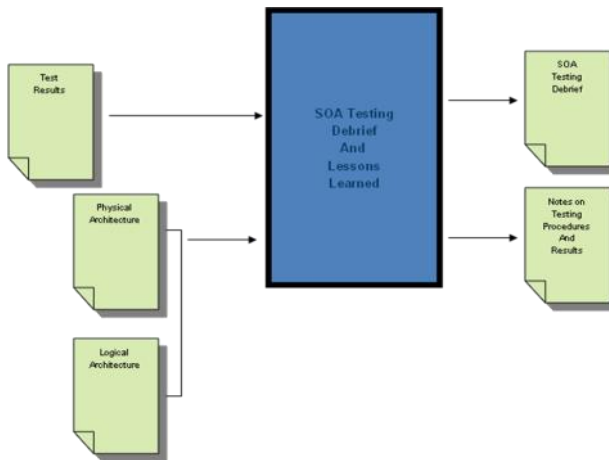
In this step we define SOA testing information for consumption by third party entities, such as the executive team. The major sub-step is:

- SOA Testing Debrief and Lessons Learned

Key artifacts created are:

- SOA Testing Debrief
- Notes on Testing Procedures and Results

**SOA testing debrief and lessons learned** means to summarize the SOA testing effort for third parties in terminology appropriate to the audience.



**Figure 22:** SOA Testing Debrief and Lessons Learned

## 19. Operational Test Planning

In this step we define operational test planning for the SOA, or the ongoing testing of the SOA during production. The major sub-steps include:

- Create Approach to Operational Test Planning
- Development of Operational Test Plan
- Select Operational Testing Tools

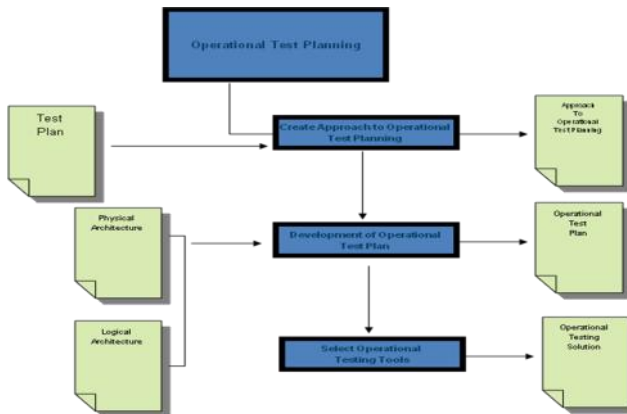
Key artifacts created are:

- Approach to Operational Test Planning
- Operational Test Plan
- Operational Testing Solution

**Create approach to operational test planning** is to create a general approach to operational testing that defines the scope, the purpose, and the guidelines for execution.

**Development of operational test plan** is to create a plan to bring operational testing to the SOA.

**Select operational testing tools** is to select the proper tools for operational test planning.



**Figure 23: Operational Test Planning**

## USING MINDREEF SOAPSCOPE SERVER



The methodology in this book was developed around the use of Mindreef SOAPscope Server™, the industry's leading solution for testing and verifying the quality of service-oriented architectures. It is server-based with hosted tools to provide always-on access so that SOA team members can perform quality-related tasks or access quality-related artifacts at any time. Every member of your project team can deliver quality SOAs and services in an agile development lifecycle, by delivering better software at each phase with:

- Enforceable SOA governance and compliance that starts with the architectural team and continues through development and testing
- Testing capabilities that let you start building quality and performance with unit tests as early as during the development phase and continue to drive it with functional, acceptance, regression, and performance tests by the testing team
- Support capabilities that allow your tech support and test teams to speed diagnosis and resolution by bundling and sharing completely reproducible test scenarios and problems with development
- Industry-leading collaboration across the entire SOA project team, that's always on and accessible from a browser

SOAPscope Server helps SOA project teams – from design through support – collaborate effectively and support the business need for agility while quickly and easily delivering well-tested, scalable, and policy compliant services and SOAs.

Mindreef SOAPscope Server Features include:

- Conduct performance and scalability testing with Mindreef Load Check™
- Author SOA compliance policies with Mindreef Policy Rules Manager™
- Collaborate across multiple roles and staff with Shared Workspaces
- Take a project focus with Service Spaces™
- Easily test outside a production environment with Named Endpoints
- Automate testing against multiple data sets with data binding
- Drive test automation with Test Suites
- Gain service understanding without XML knowledge, with Pseudocode View™
- Interact with and understand the behavior of services without needing to build a UI to drive them (Message Invoke)
- Complete testing throughout the service lifecycle with:
  - Unit Testing
  - Functional Testing
  - Acceptance Testing
  - Regression Testing
- Diagnose problems at design-time and run-time
- Point-and click test drive of services
- Achieve test-driven development and test clients and services with Simulation and Scenario Testing
- Improve WSDL understanding with Contract Overview and Documentation

SOAPscope Server is available directly from Mindreef, Inc. Mindreef Load Check and Policy Rules Manager are included with SOAPscope Server 6.0 or greater. To learn more about product licensing and service options or to request an evaluation copy, call (603) 465-2204 ext. 581, e-mail [sales@mindreef.com](mailto:sales@mindreef.com), or visit: <http://www.mindreef.com>.



## CONCLUSION



So, does testing change with SOA? *You bet it does.*

Truth-be-told, testing SOAs is a complex, disconcerting computing problem. You need to learn how to isolate, check, and integrate, assuring that things work at the service, persistence, and process layers.

The foundation of SOA testing includes selecting the right tool for the job, having a well thought out plan, and sparing no expense in testing cycles or else risk that your SOA will fail out of the gate and thus have no creditability.

Organizations are beginning to roll out their first instances of SOA, typically as smaller projects. While many work just fine, some are not living up to expectations due to quality issues that could have been prevented with adequate testing. You need to take these lessons, hard learned by others, and make sure that testing is high on your priority list when you dive into SOA.

# Discover how to:

Define your  
approach to testing

Create your  
step-by-step  
SOA test plan

Select the right  
tool for the job

Unlock the keys  
to success for  
your SOA!



SOA requires a unique approach to testing. Unless you're willing to reorient your testing procedures and technology now, you may find yourself behind the curve.

This step-by-step guide introduces you to the methodologies, processes, and key technologies that can provide a key strategic advantage to ensure your SOA will be useful and productive from the first day of operation.

**Learn More @ [www.mindreef.com](http://www.mindreef.com)**

- ✓ Download product information and whitepapers
- ✓ Attend a Webinar or listen to a podcast
- ✓ Request a free product evaluation
- ✓ Learn why thousands of customers, including 40 of the Fortune 100, use Mindreef SOAPscope products to improve SOA quality!

**Mindreef**<sup>®</sup>  
The SOA Quality Company<sup>™</sup>