

Key Strategies for SOA Testing

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

Authored by:

David S. Linthicum and
David Bressler



Key Strategies for SOA Testing

Published by Progress Software

14 Oak Park Drive, Bedford MA 01730 USA

+1-781-280-4781 or toll-free, 800-477-6473 | www.progress.com

Copyright © 2008-2009 Progress Software Corporation

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 Progress Software. Requests to the Publisher for permission should be addressed to the Legal Department, Progress Software, 14 Oak Park Drive, Bedford MA 01730 USA, or +1-781-280-4000 or toll-free, 800-477-6473.

Trademarks: Copyright © 2008-2009 Progress Software and/or its subsidiaries or affiliates. All rights reserved. Progress and Actional are trademarks or registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and other countries. Any other trademarks contained herein are the property of their respective owners.

Manufactured in the United States of America



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 ground-breaking and best selling *Enterprise Application Integration* in 1998. His latest book, *Next Generation Application Integration*, is also a best seller.

About the Authors



David Bressler is Progress Software's Actional Product and Community Evangelist. In this role, David provides customers with the expertise and experience required to deliver an enterprise service-oriented architecture (SOA) and steers technology partnerships to provide real value to customer deployments. With almost 20 years and over 30 countries of experience in enterprise software and data networking, David brings to customer engagements a deep knowledge of application integration and data networking, blended with an ability to leverage emerging technology to create business value.

David joined Progress in 2006 with the acquisition of Actional. Prior to joining Actional in 2002, David was one of the leaders responsible for bootstrapping TIBCO's European operations and the key driver behind TIBCO's relationship with Cisco. He went on to become vice president of strategic technology development at Aether Systems through their successful IPO, and then on to manage the delivery of shared middleware networks at Radianz (now BT Radianz). David received his MBA in International Business from New York University Stern School of Business, where he graduated with distinction. He can be reached at <http://twitter.com/djbressler>.

About Progress Software Corporation

Progress Software (NASDAQ: PRGS) provides application infrastructure software for the development, deployment, integration and management of business applications. Our goal is to maximize the benefits of information technology while minimizing its complexity and total cost of ownership.

Progress can be reached at www.progress.com or **+1-781-280-4000**.

Table of Contents

Introduction	1	Information Level	6
Chapter 1: SOA Testing, the Basics	2	Service Level	6
Service Distribution	2	Process Level	7
Service Data	2	Bringing the Plan Together	8
Service Behavior	2	Chapter 3: Step-by-step Guide to SOA Testing	13
Service Integrity	2	So, How Do You Go about Testing a SOA?	13
Service Design	3	1. Define Testing Domain	14
SLAs and Performance Testing	3	System Description Analysis	
Adherence to a Holistic Architecture	3	SOA Testing Proof of Concept	
Chapter 2: Creating a SOA Test Plan	4	2. Define Architectural Objectives	15
Understanding Your Own Needs	4	Define Logical Architecture	
Defining the Approach	4	Define Physical Architecture	
Understanding the Domain	5	3. Design Review and Test Planning	16
Considering Service Performance	5	Review Core Design	
Testing Levels	6	Create Test Plan	
		Create Test Plan and Implementation Schedule	

4. Create Functional Testing Approach 17

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

5. Define Performance Requirements 18

- Create Service-level Performance Requirements
 - Create Data-level Performance Requirements
 - Create Process-level Performance
-

6. Define SLA Requirements 19

- Create Service-level SLA
 - Create Data-level SLA
 - Create Process-level SLA
-

7. Define Data Layer Testing Approach 20

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

8. Define Services Layer Testing Approach 21

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

9. Define Policy Layer Testing Approach 22

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

10. Define Process Layer Testing Approach 23

- Create Process Layer Definition
 - Create Process Testing Approach
-

11. Define Service Simulation 24

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

12. Create Core Scenarios 25

- Create Approach to Scenarios
 - Create Specific Scenarios
-

13. Creating User-defined Compliance Rules 26

- Create Approach to Compliance Rules
 - Create Compliance Rules
-

14. Select Your SOA Testing Technology Suite 27

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

15. Testing Execution 28

- Unit Testing
 - Functional Testing
 - Regression Testing
 - Compliance and Validation
-

16. Looping Back to Design and Development 29

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

17. Define Diagnostics for Design-time and Run-time 30

18. SOA Testing Debrief and Lessons Learned 31

19. Operational Test Planning 32

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

Chapter 4: Using Actional Application Development 33

Conclusion 34



Introduction

In this book we refer to **SOA** or **service-oriented architecture** in the larger context of services. Not just those services confined to the SOAP and WSDL specifications. 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 Actional Team Server, an important component of Progress® Actional® Application Development and Progress® Actional® Enterprise, that when used with the SOA testing approaches defined in this paper, will provide a strategic advantage through automation and repeatability.



Chapter 1: 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 services 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. Service distribution comes with its own set of challenges, including the ability to discover the services under test in heterogeneous environments. Moreover, the actual run-time testing of each service is complex in itself.

Service Data

Testing of **the data** as it flows through a service is critical to the health of the service. While many people 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 include 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, as well as 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, and mixing them with different security context. Establishing coverage metrics is critical here. Moreover, if the service is designed to provide different behavior depending on context, that needs to be monitored throughout the processing of the service as well. The approach is the same: 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 of 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, the design patterns used, and the ability of that service to live up to that particular design. This process involves interaction between multiple roles, such as architects, designers, and developers.

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

SLAs and Performance Testing

Service-level agreements (SLAs) are internal or external contracts that define the consistent performance of a service. The goal 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 the level of quality.

Adherence to a Holistic Architecture

Finally, it's important to note that a SOA is an architecture. As a result, 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, through data and transactional services, to orchestration and composite application development.

Chapter 2: Creating a SOA Test Plan

Creating a test plan for a SOA is more involved than you might think. You need 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, as a linked-group, and holistically as an architecture.

One of the things to consider is enterprise- and project-wide test planning. Typically, this involves 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 project, the ability of the projects to drive their issues up to the enterprise-wide testing strategy, and the ability of 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 there are differences 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: These are SOA architectures in which the use of transactional services is more apparent. Typically, these are on-line transaction-processing types of application clusters that use an architecture in which transactional services are leveraged and invoked more than others.

Data Heavy: These are SOA architectures in which most of the services employed are data services or services that broker information more than behavior.

Process Heavy: These are SOA architectures in which the core dynamics are driven at the process level. Typically, in these architectures 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 presented here we call out steps to define the approach to testing for each component. Prior to that, and within the plan, we need to focus on how we're going to approach testing holistically.

There are a few approaches for testing SOA, including:

- > Bottom Up
- > Top Down
- > System

The **bottom-up** approach looks to test the SOA from the most primitive features to the most sophisticated. Typically, this works up from the data to the process (see Figure 1), moving through the data-services, data-abstraction, and transactional-services layers to the process layers, and, finally, to the monitoring and event management layers. This is done by testing each component at the lower layers, then 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, then moving down to the lower layers.

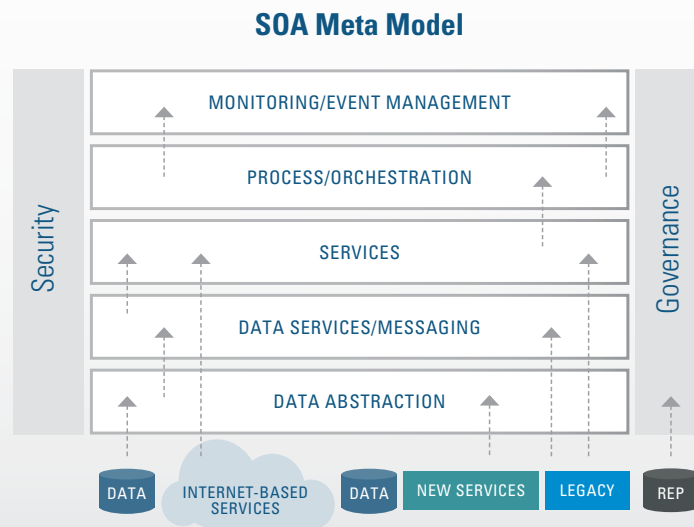


Figure 1: To approach SOA testing you can work up from the bottom, down from the top, test components or the entire architecture.

The **system** approach tests the architecture holistically, looking at the entire solution set as one functional unit 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.

Understanding the Domain

When you create a test plan, your primary goal 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 understanding comes from the analysis done when the architecture was created.

Considering Service Performance

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

- > Defined at a high level by the approach
- > Defined at a low level by 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 or overall SOA performance 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 there are three levels:

- > Information Level
- > Service Level
- > Process Level

Information Level

The information level of a SOA refers to the underlying data that links to the services. While in many cases the data sources are standard databases, there are also legacy systems, CRM systems, ERP systems, and other "stovepipe" applications that 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, that is important, but also it's the use of the data within the services.

You test data within services by 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. In fact, all services are abstracted into the business process or orchestration layer, which provides the agile nature of a SOA because you can create and change solutions by stitching together the services into a new configuration.

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: the services must be able to properly function both by themselves and 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 using testing harnesses or SOA testing tools (discussed later). You also need to consider any services that the service may employ and, thus, test these related services holistically as a single logical service. In some cases you may be testing a service that calls a service in which some of the services are developed and managed in-house and some exist on remote systems 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 that 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 uses 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 up" the services at a live HTTP endpoint.

Services should have the appropriate granularity. Don't focus on too fine-grained or too coarse-grained services. Focus on the correct granularity for the purpose and use within the SOA. Here the issues being tested involve performance more 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 services don't provide the proper autonomic values to support their reuse. You need to work with the service designer on this issue.

Process Level

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.

You 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 seriousness as you test 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 creating a process, and also drives sequencing, nesting, and managing 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. Using BPEL for services orchestration is an example of an approach; choreography and proprietary process engines are others. By respecting the service interface boundary, the service implementation is largely irrelevant. The SOA tenets create constrained and verifiable layers in the architecture.

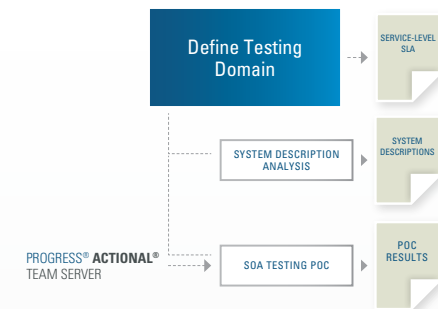
Bringing the Plan Together

Creating the core plan for SOA testing means considering all of the issues 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.

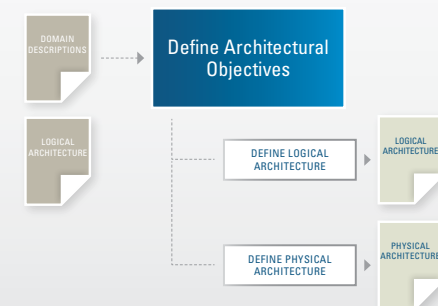
A common pattern in all SOA is testing services. SOA testing boils down to testing services. Thus, testing services as collections, as stand-alone, or holistically as a complete architecture is essential. Equally important are how you create a plan and what testing technology you use to test services for 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:

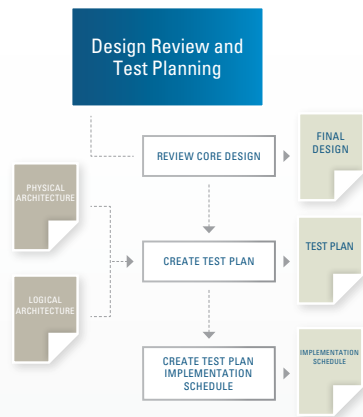
1. Define the Testing Domain



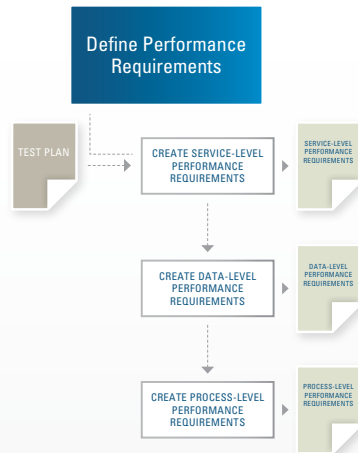
2. Define Architectural Objectives



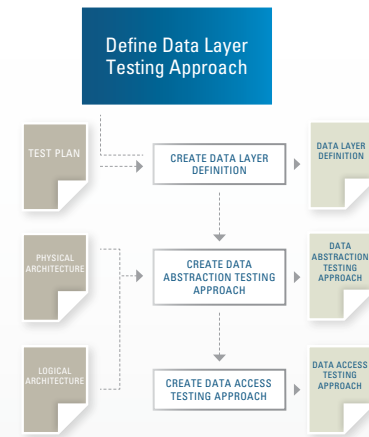
3. Design Review and Test Planning



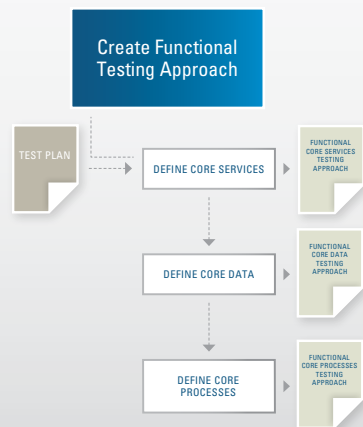
5. Define Performance Requirements



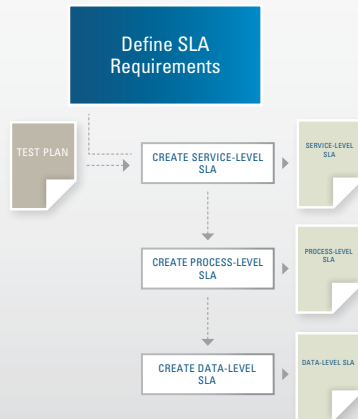
7. Define Data Layer Testing Approach



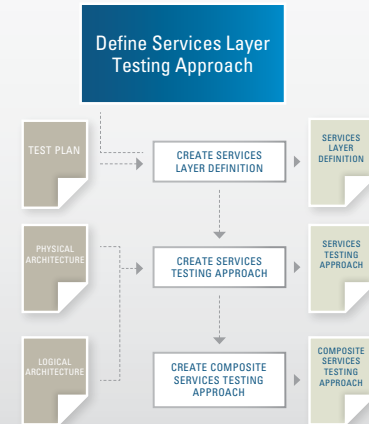
4. Create Functional Testing Approach



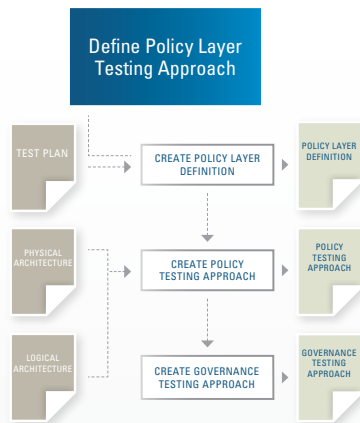
6. Define SLA Requirements



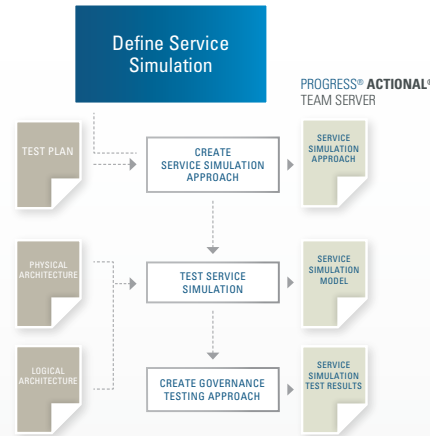
8. Define Services Layer Testing Approach



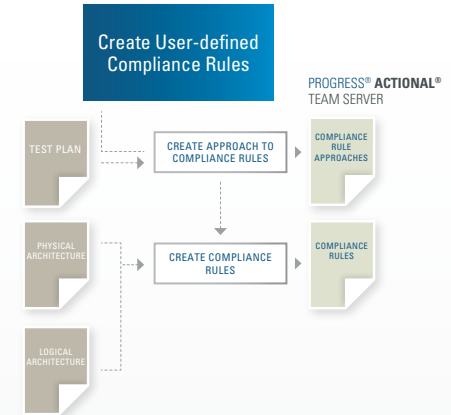
9. Define Policy Layer Testing Approach



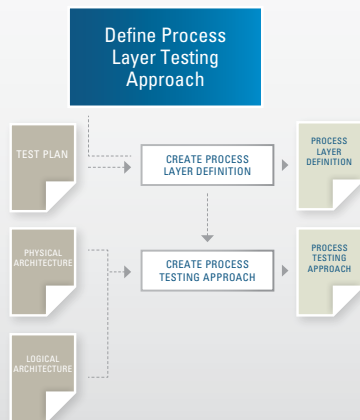
11. Define Service Simulation



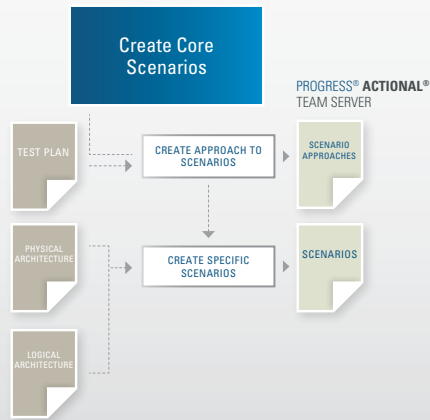
13. Creating User Defined Compliance Rules



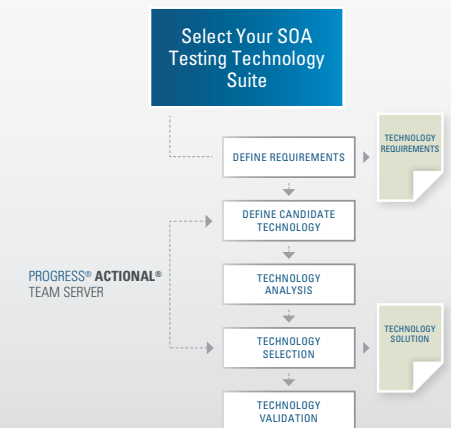
10. Define Process Layer Testing Approach



12. Create Core Scenarios

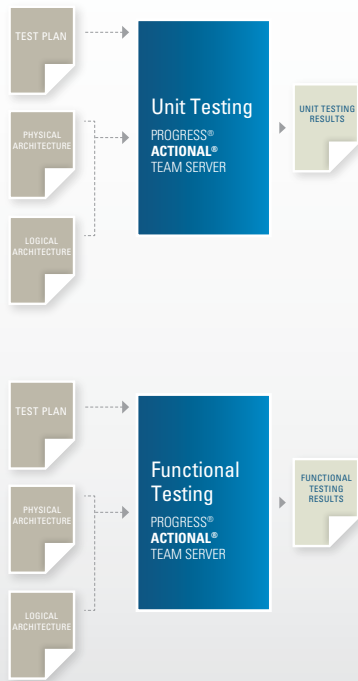


14. Select SOA Testing Technology Suite

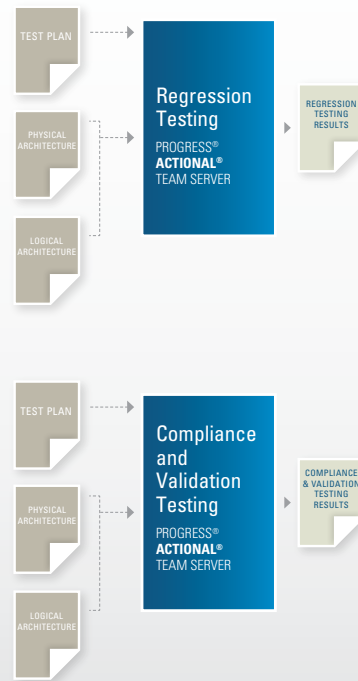


15. Testing Execution

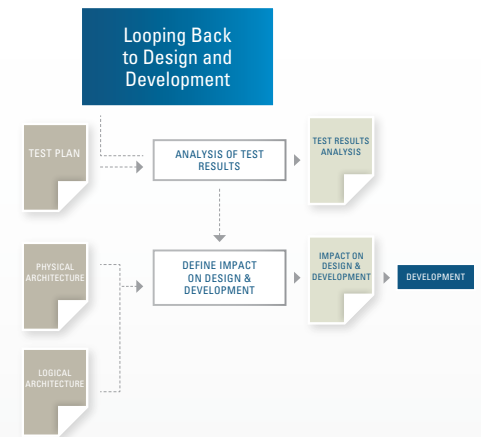
- > Unit Testing
- > Functional Testing



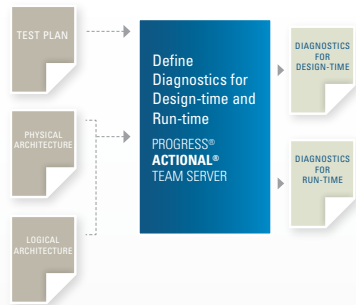
- > Regression Testing
- > Compliance and Validation (WSDL, Schema, Messages)



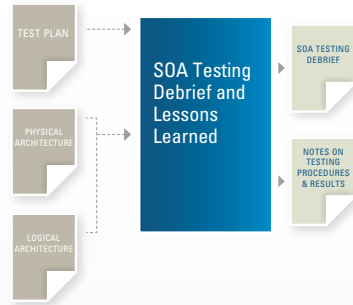
16. Looping Back to Design & Development



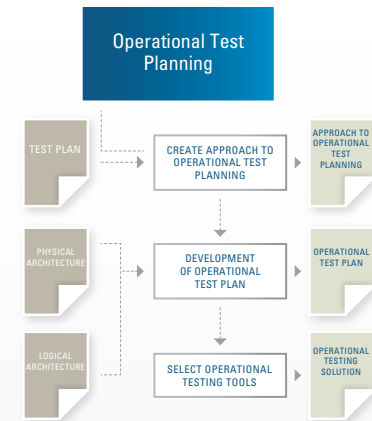
17. Define Diagnostics for Design-time & Run-time



18. SOA Testing Debrief & Lessons Learned



19. Operational Test Planning Approach



Chapter 3: Step-by-step Guide to SOA Testing

So, How Do You Go about Testing a SOA?

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

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

- > This methodology is iterative. 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. 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. Customize it for your purposes, as needed; even add or change steps.

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

SOA testing proof of concept involves leveraging a testing tool within the POC. This is where the Progress® Actional® family of products first comes into play as you determine how to leverage the tools in a SOA. The data points resulting from the POC will provide critical information on how the final SOA testing will be defined.

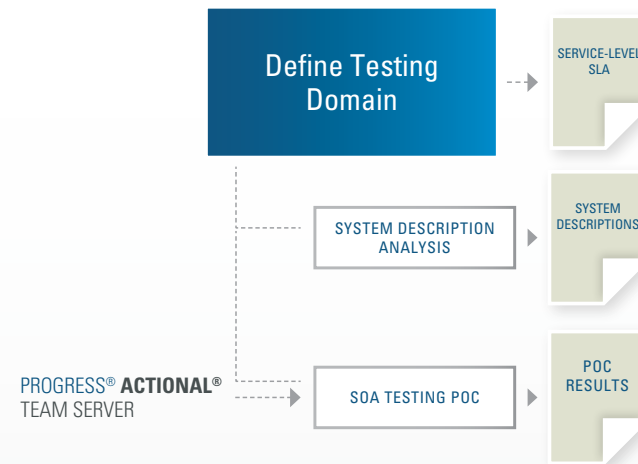


Figure 2: Define Testing Domain

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 consists of defining how the architecture exists conceptually, independent of physical instances of technology. The resulting document provides an understanding of the logical components of the SOA and how each component interacts with others.

Define physical architecture focuses on how the physical instances of technology (ESBs, orchestration, service platform) interact, one with another. Like the logical architecture, this document demonstrates how all of the components interact, but now defines all components, interfaces, standards, and enabling technologies. Both the logical and physical architectures are critical to the remainder of the process.

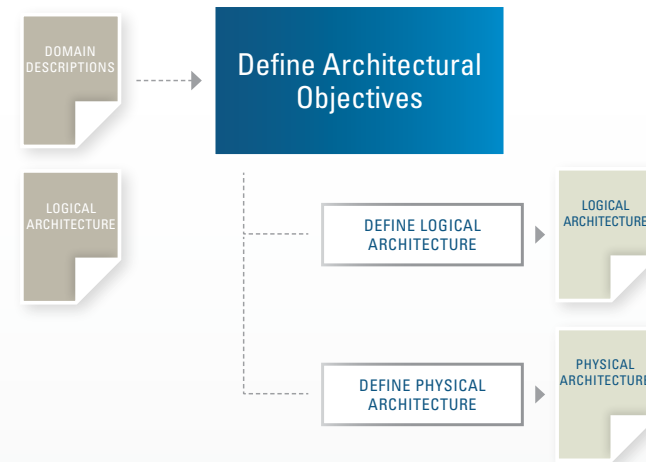


Figure 3: Define Architectural Objectives

3. Design Review and Test Planning

In this step we set up the testing project, including reviewing the core design of the SOA, both its logical and physical architectures, and determine how we are going to approach testing, the test plan to use, 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 discussed above, involves 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 planning before reviewing the core design.

Create test plan focuses on leveraging the design and architecture information to put together an approach and plan for testing the SOA, including all processes, procedures, and enabling technologies. The resulting test plan is a foundation document for the rest of the methodology.

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

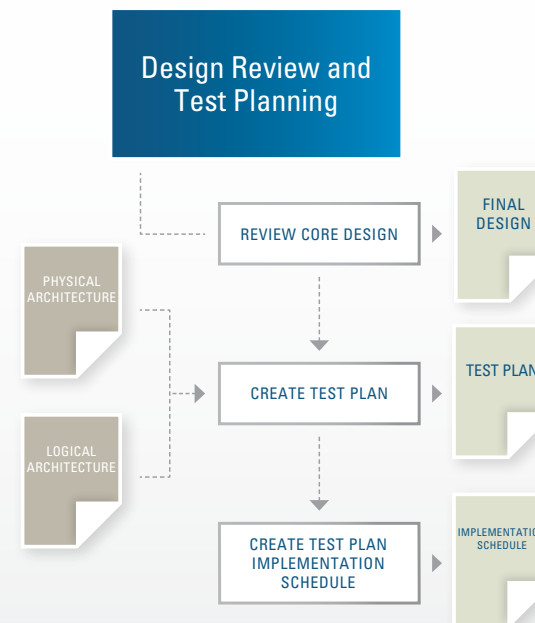


Figure 4: Design Review and Test Planning

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 entails listing and defining all of the services within the problem domain that will be under test and the approach for functionally testing those services. This involves understanding what they do, whom they are for, and what the native metadata are, etc.

Define core data requires listing and defining all of the data within the problem domain that will be under test and the approach for functionally testing that data.

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

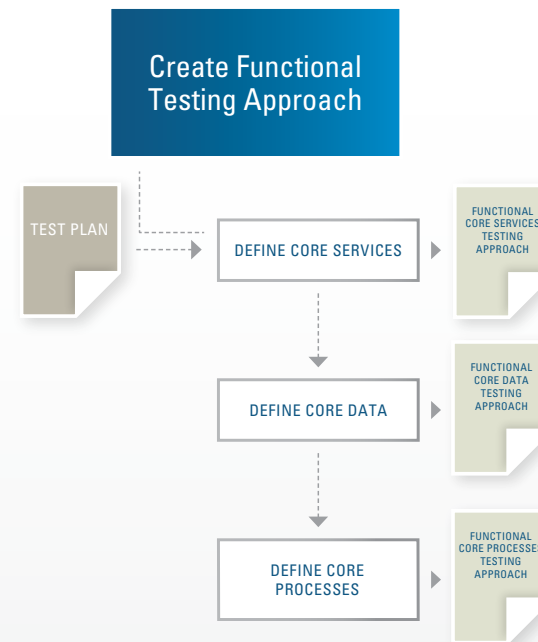


Figure 5: Create Functional Testing Approach

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 focuses on determining performance expectations for all services within the testing domain, including response time under an increasing load.

Create data-level performance requirements consists of determining performance expectations for all data access services, including response time under an increasing load.

Create process-level performance requirements entails determining performance expectations for all processes, including response time under an increasing load.

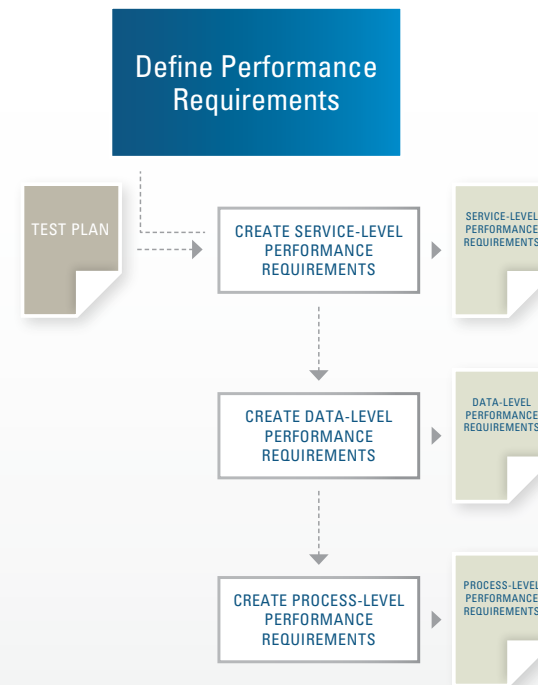


Figure 6: Define Performance Requirements

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 requires defining the performance expectations and agreement for particular services, including response times and uptime requirements.

Create data-level SLA involves defining the performance expectations and agreement for particular data (persistence), including response times and uptime requirements. This sub-step reveals that certain data is more important than other data and should be tested to a level that is commensurate with the importance of the data.

Create process-level SLA focuses on defining the performance expectations and agreement for particular processes, including response times and uptime requirements.

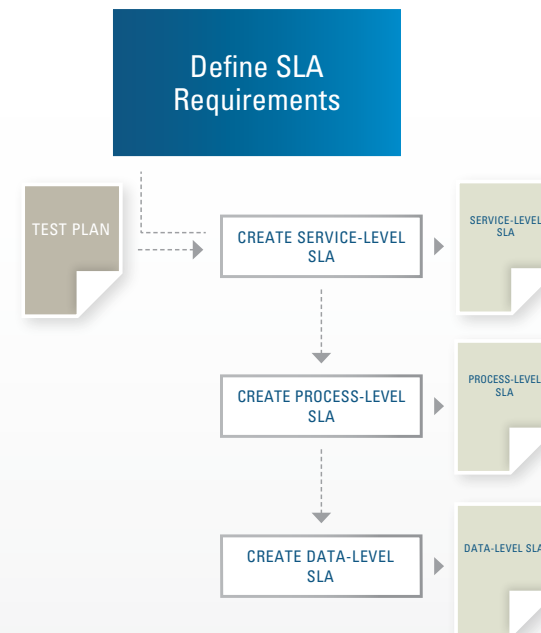


Figure 7: Define SLA Requirements

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

Create data abstraction testing approach involves defining any data abstractions in place and the approach for testing the data abstractions that have remapped the physical database to different virtual structures.

Create data access testing approach consists 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.

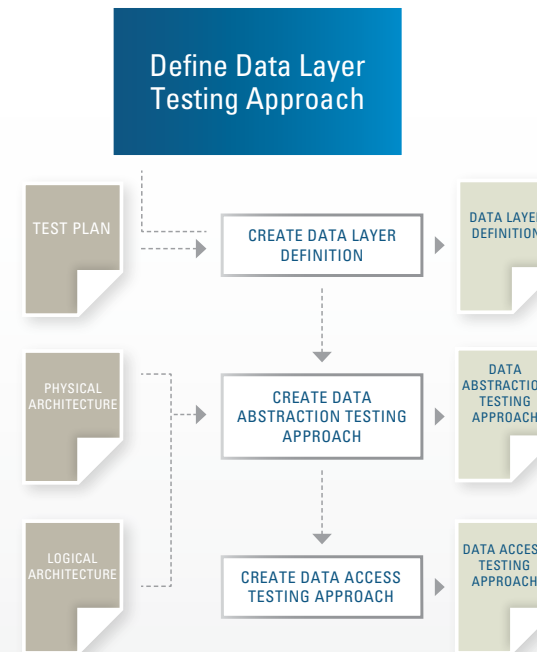


Figure 8: Define Data Layer Testing Approach

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 consists of defining all services that are under test, including access approaches, enabling technology, and use of standards.

Create services testing approach focuses on determining the best way to approach testing the services defined in the previous step. It requires 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, focuses on determining the best way to approach testing of the composite services. This requires understanding access mechanisms, service grouping, and other information that will be useful in creating the approach.

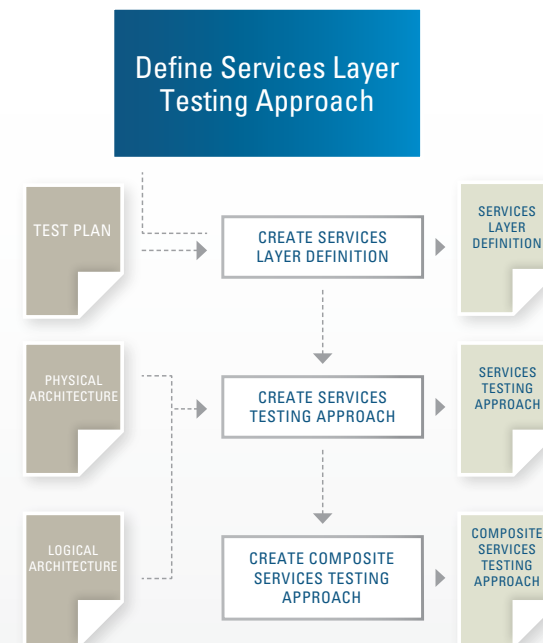


Figure 9: Define Services Layer Testing Approach

9. Define Policy Layer Testing Approach

In this step we define the policy layer testing approach. Most companies that are building Web services encounter interoperability issues when services are implemented and consumed using tools from different vendors. To mitigate this problem, 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 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 consists 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, they need to be tested to determine that they are functioning per the design.

Create policy testing approach entails 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 involves determining the best way to approach testing of the holistic governance layer.

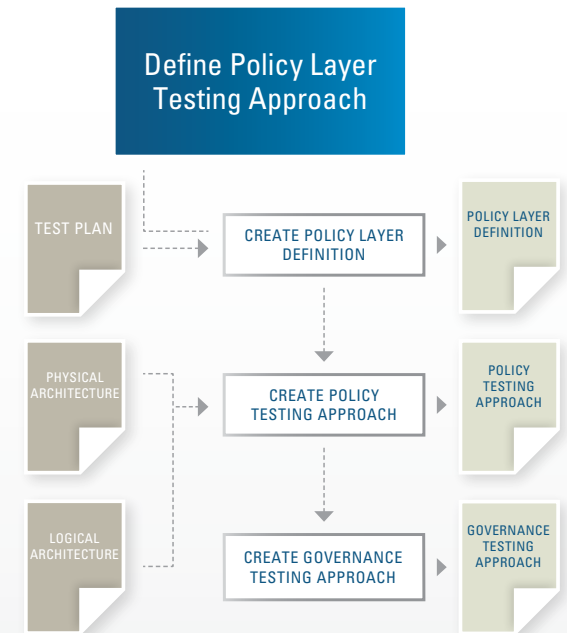


Figure 10: Define Policy Layer Testing Approach

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 involves defining all processes that are under test (i.e., what they are, who owns them, and how they are designed), including access approaches, enabling technology, and use of standards.

Create process testing approach focuses on determining the best way to approach testing the process defined in a previous step. It requires understanding access mechanisms, process grouping, and other information that will be useful in creating the approach.

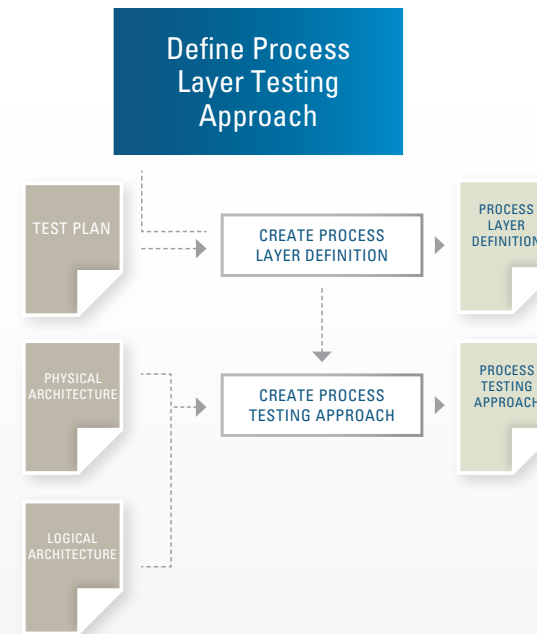


Figure 11: Define Process Layer Testing Approach

11. 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. Simulated services provide a mechanism to test the services and SOA prior to deployment and when a services is not available.

Create service simulation model takes the approach created in the previous step and defines the model for use while testing.

Test service simulation consists of using the service simulation approach and service simulation model to test the simulated services. Any issues are noted here and addressed in this step.

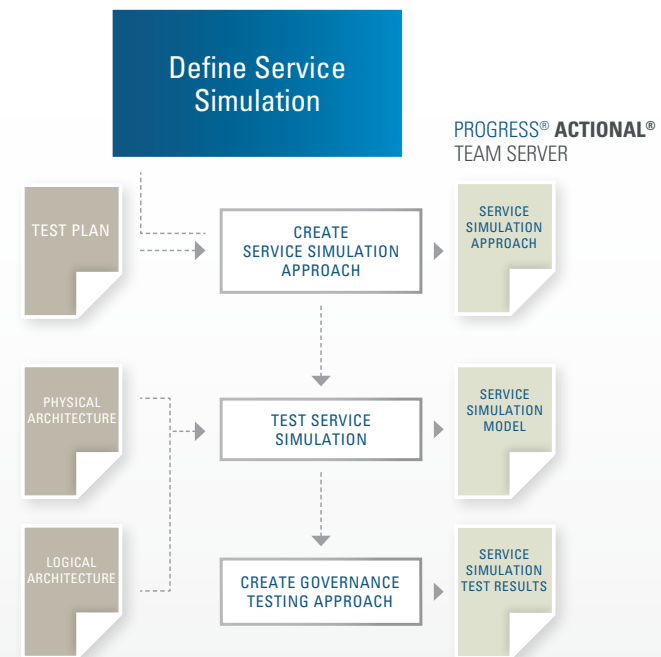


Figure 12: Define Service Simulation

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 involves creating a general approach to the scenarios or use cases employed to test the SOA. These scenarios 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.

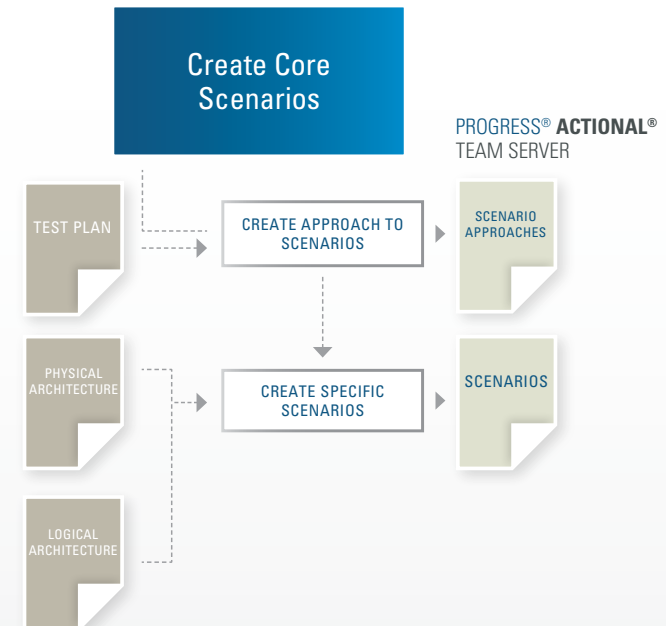


Figure 13: Create Core Scenarios

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 in the next step.

Create compliance rules entails formulating the actual compliance rules for the SOA, leveraging the approach created in the previous step.

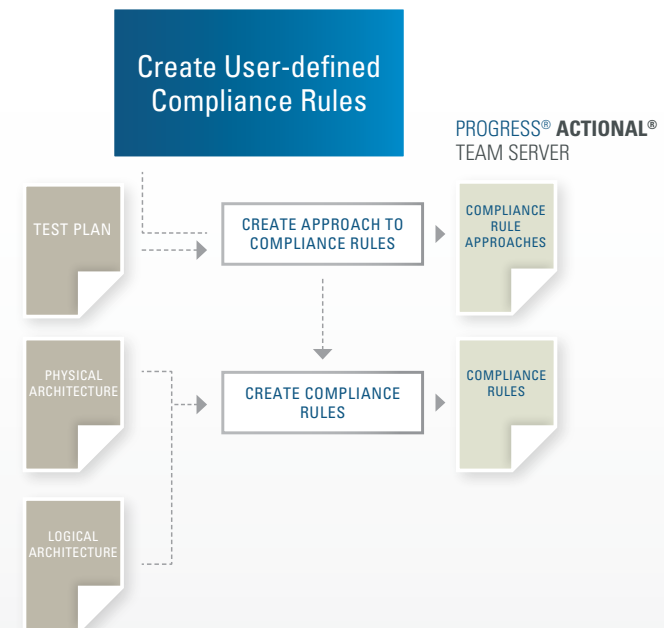


Figure 14: Create User-defined Compliance Rules

14. Select Your 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 formulates the technology to be employed, including the service testing suite and other technology required for SOA testing.

Define candidate technology involves listing the technology required for SOA testing, including integrated software solutions such as Actional Application Development.

Technology analysis involves considering the technology for fit and function, for the application to the SOA testing procedures defined in the previous step.

Technology selection involves choosing Actional testing technology to test the SOA with the artifact previously defined.

Technology validation entails validating 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.

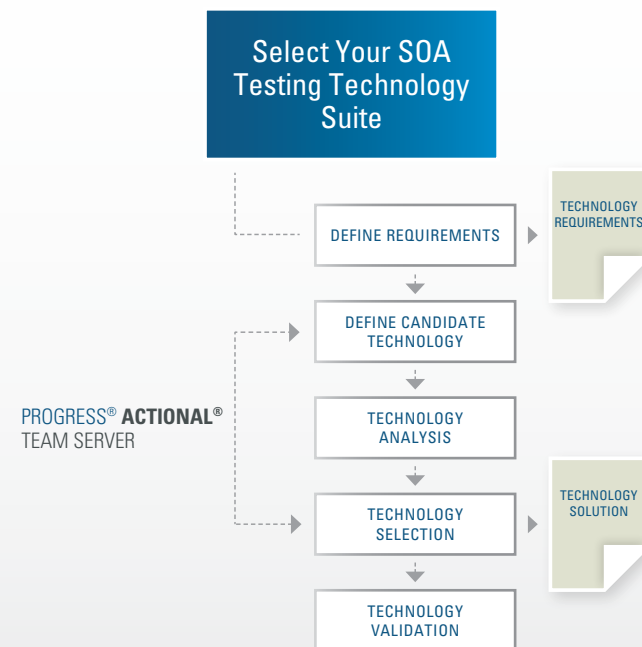


Figure 15: Select Your SOA Testing Technology Suite

15. Testing Execution

In this step we carry out SOA testing, including:

- > Unit Testing
- > Functional Testing
- > Regression Testing
- > Compliance and Validation (WSDL, Schema, Messages)

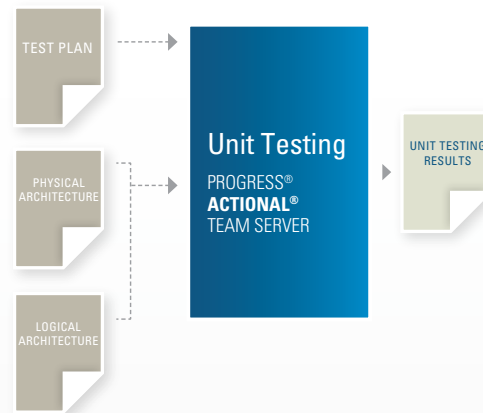


Figure 16: Unit Testing

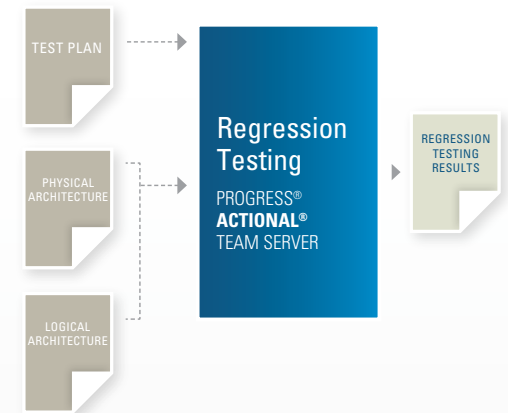


Figure 18: Regression Testing

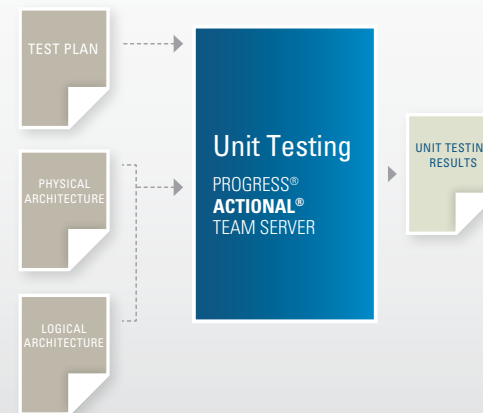


Figure 17: Functional Testing

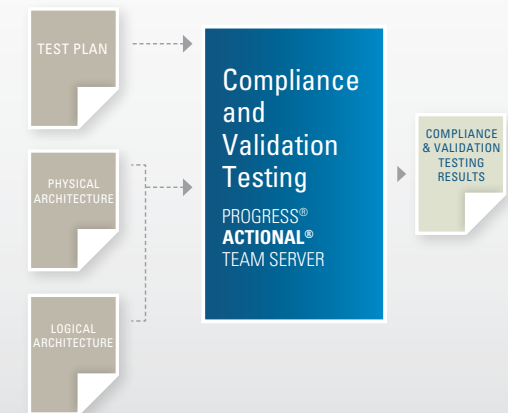


Figure 19: Compliance and Validation Testing

16. Looping Back to Design and 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 focuses on looking at the test results of the services/SOA testing process defined in this methodology and preparing the results for feedback to design and development.

Define impact on design and development involves sending the results back from the services/SOA testing process to development.

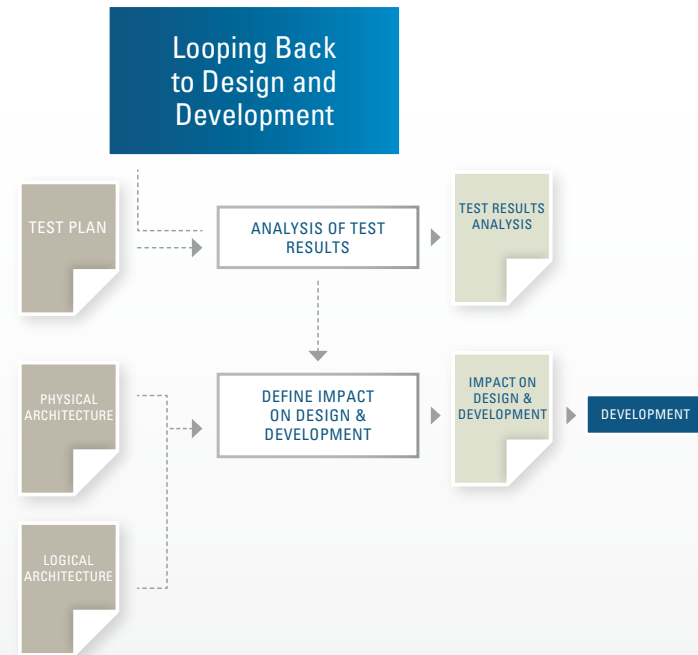


Figure 20: Looping Back to Design and Development

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 involves creating the diagnostic approaches for the design-time and run-time instances of the SOA.

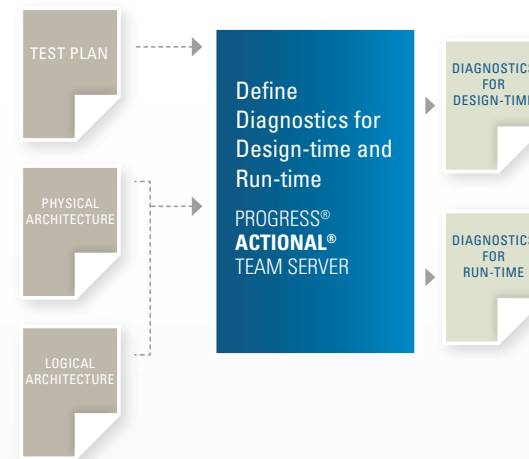


Figure 21: Define Diagnostics for Design-time and Run-time

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 summarizes the SOA testing effort for third parties in terminology appropriate to the audience.

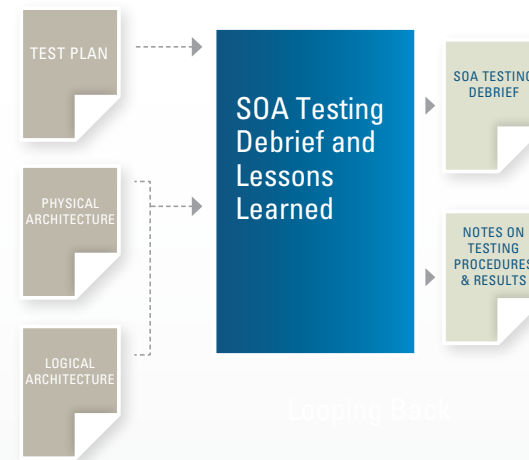


Figure 22: SOA Testing Debrief and Lessons Learned

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 focuses on formulating a general approach to operational testing that defines the scope, purpose, and guidelines for execution.

Development of operational test plan entails creating a plan to bring operational testing to the SOA.

Select operational testing tools involves choosing the proper tools for operational test planning.

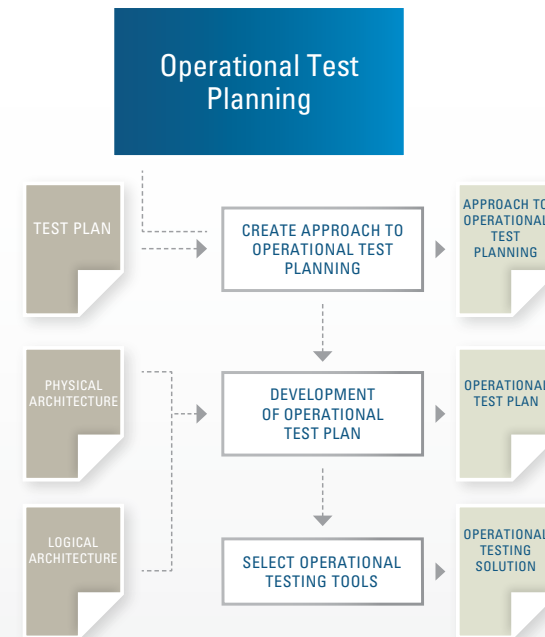


Figure 23: Operational Test Planning

Chapter 4: Using Actional Application Development

The methodology in this book was developed around the use of Actional Application Development, the industry's leading solution for testing and verifying the quality of service-oriented architectures. Actional Application Development 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 the 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 them 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
- > End-to-end transaction visualization without coding that enables developers to optimize services by seeing how application components are put together and work

Actional Application Development 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.

Actional Application Development features include:

- > Conduct performance and scalability testing with Load Check™
- > Author SOA compliance policies with 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)
- > See end-to-end transaction path with Flow Mapping™ to fix potential problem areas in production
- > Complete testing throughout the service lifecycle with:
 - Unit testing
 - Functional testing
 - Acceptance testing
 - Regression testing

- > Diagnosis of problems at design-time
- > Point-and click test drive of services
- > Test-driven development and testing clients and services with simulation and scenario testing
- > Improved WSDL understanding with contract overview, annotation, and documentation

To learn more about product licensing and service options or to request an evaluation copy, call **+1-781-280-4000** or toll-free, **800-477-6473**, or visit: **www.progress.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 usually begin 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

- > **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 Progress Actional Application Development to improve SOA quality!

www.progress.com

PROGRESS
SOFTWARE

