SOA Testing using Black, White and Gray Box Techniques

By Mamoon Yunus and Rizwan Mallal, Crosscheck Networks

Black, White & Gray Box SOA Testing is essential for deploying robust, scalable, interoperable and secure Web Services.

I. Introduction

Web Services are the foundations of modern Service Oriented Architecture (SOA). Typical Web Services include message exchange between a consumer and a producer using SOAP request and responses over the ubiquitous HTTP protocol. A Web service producer advertises its services to potential consumers through Web Services Description Language (WSDL) – an XML file that contains details of available operations, execution endpoints and expected SOAP request-response structures.

Many testing techniques and methodologies developed over the years apply to Web Services-based SOA systems as well. Through functional, regression, unit, integration, system and process level testing, the primary objective of testing methodologies is to increase confidence that the target system will deliver functionality in a robust, scalable, interoperable and secure manner.

Techniques such as **Black**, **White** and **Gray** Box testing applied to traditional systems map well into Web Services deployments. However, the following characteristics of a Web Services deployments introduce unique testing challenges:

- Web Services are intrinsically distributed and are platform and language agnostic.
- Web Services can be chained with dependencies on other 3rd party Web Services that can change without notice.
- Web Services ownership is shared across various stakeholders.
- Web Services client developers typically only have access to interfaces (WSDLs) and lack access to code.

In this paper, we will investigate testing techniques and their application to Web Services. We will use a simple sample Web service to illustrate each of these techniques and the relative strengths and weaknesses of such techniques. Finally, a novel approach that extends Gray Box's reach into realm of White Box testing by leveraging the rich information provided in the WSDL file will be described.

Key Concepts

Essential Testing Techniques
Using Black, White, & Gray Box Testing techniques are key to deploying robust SOA.

Gray Box Testing is ideal for Web Services Consumers
Published Web Services operations in WSDLs provide rich enough information for consumers to derive benefits of Gray Box Testing.

Extending Gray Box's Reach into the Domain of White Box Testing
Using Web Services WSDLs, intelligent auto-generated tests can elevate the Gray Box Testing closer to White Box Testing without the associated burdens.

Auto-Generated Tests
Mutations based on Web Services WSDLs enable users to auto-generate tests that exercise multiple code-paths extensively in published operations.

II. Black, White and Gray Box Testing for Web Services

A. Black Box Testing

Definition: Black Box testing refers to the technique of testing a system *with no knowledge* of the internals of the system. Black Box testers do not have access to the source code and are oblivious of the system architecture. A Black Box tester typically interacts with a system through a user interface by providing inputs and examining outputs without knowing where and how the inputs were operated upon. In Black-Box testing, target software is exercised over a range of inputs and the outputs are observed for correctness. How those outputs are generated or what is inside the Box doesn't matter to the tester.

Web Service Example: To illustrate Black Box testing for a sample Web Services, an operation **Divide** that simply divides to integers *a* and *b*. The Black Box tester is unaware of what operating system, programming language, 3rd party libraries or other Web Services are being used under the hood to perform the **Divide** operation.

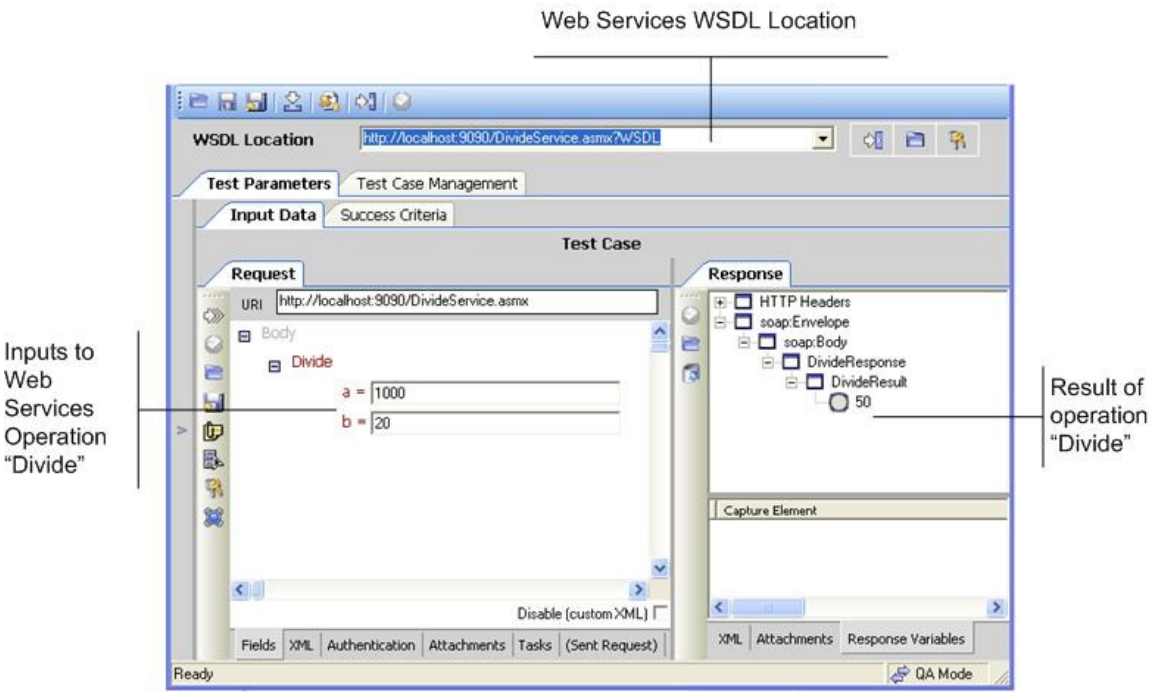

Figure 1: Black Box Testing Example for Divide Operation.

As shown in **Figure 1**, the Black Box tester has the ability to insert inputs to the operation and look at outputs. The tester may know that the Web Services WSDL is located but is completely oblivious of implementation details, program execution states, and internal exception handling. The tester has a specification and goes through a rigorous, time-consuming and oftentimes redundant exercise of trying values and ensuring that the operation **Divide** functions as expected.

Advantages

- *Efficient Testing* – Well suited and efficient for large code segments or units.
- *Unbiased Testing* – clearly separates user's perspective from developer's perspective through separation of QA and Development responsibilities.
- *Non intrusive* – code access not required.
- *Easy to execute* – can be scaled to large number of moderately skilled testers with no knowledge of implementation, programming language, operating systems or networks.

Disadvantages

- *Localized Testing* – Limited code path coverage since only a limited number of test inputs are actually tested.
- *Inefficient Test Authoring* – without implementation information, exhaustive input coverage has unknown additional benefits to the actual code paths exercised and can require tremendous resources.
- *Blind Coverage* – cannot control targeting code segments or paths which may be more error prone than others.

Black Box testing is best suited for rapid test scenario testing and quick Web Service prototyping. This testing technique for Web Services provides quick feedback on the functional readiness of operations through quick spot checking. Black Box testing is also better suited for operations that have enumerated inputs or tightly defined ranges or facets so that broad input coverage is not necessary.

B. White Box Testing

Definition: White Box testing refers to the technique of testing a system *with knowledge* of the internals of the system. White Box testers have access to the source code and are aware of the system architecture. A White Box tester typically analyzes source code, derives test cases from knowledge about the source code, and finally targets specific code paths to achieve a certain level of code coverage.

Web Service Example: To illustrate White Box testing, Figure 2 presents a simple set of rudimentary operations in C#. The first Operation **Divide** takes in 2 integers and returns *a* divide-by *b* with no checks on the input values. The second operation **safeDivide** takes in 2 string parameter inputs, but in contrast to the first operation **Divide**, the **safeDivide** operation has a broad exception handling mechanism that safely catches errors for cases where either bad data types of bad data values are sent.

A White Box tester with access to such details about both operations can readily craft efficient test cases that exercise boundary conditions. Just by observing the code, a White Box tester can immediately try:

- Divide-by-Zero scenario by setting the denominator *b* to zero
- Integer Overflow scenario by setting either integer a value > ± 2,147,483,647
- Orthogonal Data types for example floats, date, decimal data types.
- Special characters.

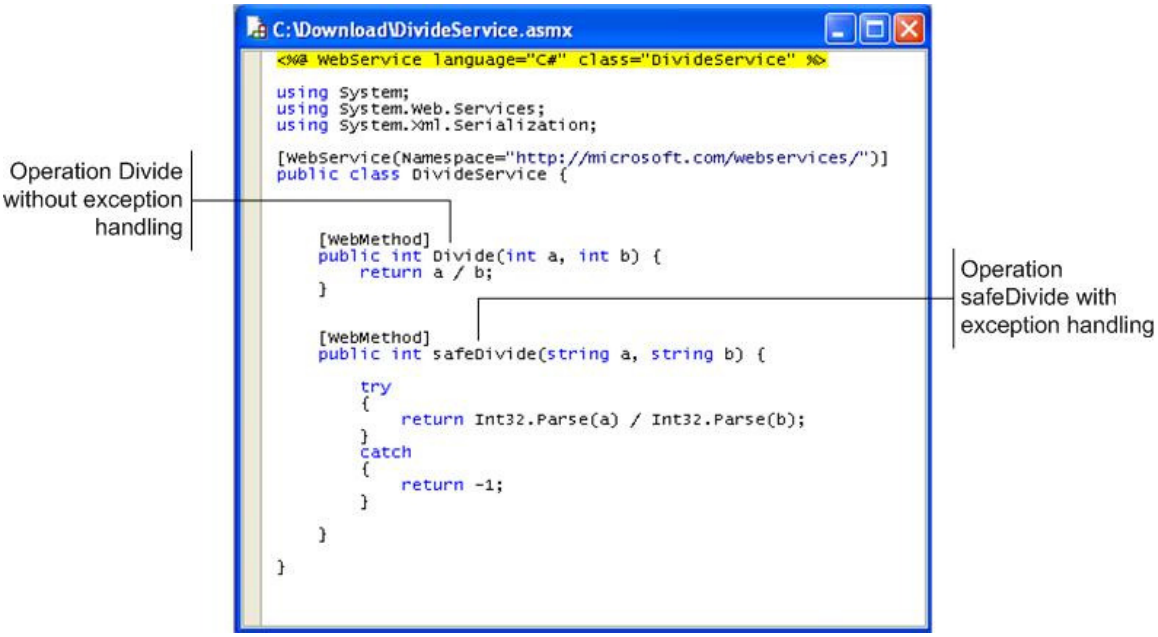

Figure 2: White Box Testing Example for 2 Web Services Operations.

A simple Divide-by-Zero test, executed for the **safeDivide** operation returns a rudimentary error code -1 as expected. However, performing the same Divide-by-Zero test on the **Divide** operation results in the following verbose stack trace in the SOAP Response:

```
1. <soap:Fault>
2.     <faultcode>soap:Server</faultcode>
3.     <faultstring>System.Web.Services.Protocols.SoapException: Server was unable to process
       request. --- System.DivideByZeroException: Attempted to divide by zero. at
       DivideService.Divide(Int32 a, Int32 b)End of inner exception stack trace ---
4.     </faultstring>
5. </soap:Fault>
```

In the **Divide** operation, a White Box tester can quickly identify and subsequently verify that the target operation has no custom exception handling but rather relies on the vendor container running the Web Service to handle the exception. The tester would then point this lack of exception handling to the

development teams and bring this operation inline with the **safeDivide** operation where *try-catch* flow is used for exception handling. A Black Box tester may perhaps identify the weakness through blind testing, however, the level of effort and the number of iterations would be large and the probability of stumbling on to such defects would be low, especially with increasing program complexity.

Advantages

- *Increased Effectiveness* – Crosschecking design decisions and assumptions against source code may outline a robust design, but the implementation may not align with the design intent.
- *Full Code Pathway Capable* – all the possible code pathways can be tested including error handling, resource dependencies, and additional internal code logic/flow.
- *Early Defect Identification* – Analyzing source code and developing tests based on the implementation details enables testers to find programming errors quickly.
- *Reveal Hidden Code Flaws* – access to source code improves understanding and uncovering unintended hidden behavior of program modules.

Disadvantages

- *Difficult To Scale* – requires intimate knowledge of target system, testing tools and coding languages, and modeling. It suffers for scalability of skilled and expert testers.
- *Difficult to Maintain* – requires specialized tools such as source code analyzers, debuggers, and fault injectors.
- *Cultural Stress* – the demarcation between developer and testers starts to blur which may become a cultural stress.
- *Highly intrusive* – requires code modification has been done using interactive debuggers, or by actually changing the source code. This may be adequate for small programs; however, it does not scale well to larger applications. Not useful for networked or distributed systems.

White-Box testing is most suited for Web Services early in the development cycle where the developer and the tester may collaborate to identify defects. White-Box testing is problematic for large SOA deployments where the distributed nature of services makes it easy for 3rd party Web Services to be invoked from within other Web Services. This results in the lack of knowledge of programming language, operating systems and hardware platforms. Unlike calling functions from a shared library running in the same memory space, distributed Web Services provide additional access challenges making White-Box testing across a SOA next to impossible.

C. Gray Box Testing

Definition: Gray Box testing refers to the technique of testing a system with *limited knowledge* of the internals of the system. Gray Box testers have access to detailed design documents with information beyond requirement documents. Gray Box tests are generated based on information such as state-based models or architecture diagrams of the target system.

Web Service Example: To illustrate Gray Box testing, **Figure 3** presents a Web Services Definition Language (WSDL) file for the simple **Divide** and **safeDivide** operations. Lines 3-36 show the data types for the messages. These lines as well as Lines 23-24 and Lines 43-45 point to **safeDivide** request message using unbounded strings. Even without access to source code or binaries, this would indicate to a tester to try buffer overflow type boundary conditions.

Without access to source code or binaries, a web service tester can only consume and invoke Web Services through WSDL files. With a rich array of information available through such WSDLs, and the inability to modify code or binaries for White Box testing, a Web Services tester can use details such as the location of the Web Service and the transport protocol (Line 82), data types (Lines 3-36), etc. provide significant leverage for authoring intelligent, efficient and highly target test cases.

Figure 3: WSDL file (partially collapsed) for **Divide** and **safeDivide** operations.

Advantages

- *Offers Combined Benefits* – Leverage strengths of both Black Box and White Box testing wherever possible.
- *Non Intrusive* – Gray Box does not rely on access to source code or binaries. Instead, based on interface definition, functional specifications, and application architecture.
- *Intelligent Test Authoring* – Based on the limited information available, a Gray Box tester can author intelligent test scenarios, especially around data type handling, communication protocols and exception handling.
- *Unbiased Testing* – The demarcation between testers and developer is still maintained. The handoff is only around interface definitions and documentation without access to source code or binaries.

Disadvantages

- *Partial Code Coverage* – Since the source code or binaries are not available, the ability to traverse code paths is still limited by the tests deduced through available information. The coverage depends on the tester authoring skills.
- *Defect Identification* – Inherent to distributed application is the difficult associated in defect identification. Gray Box testing is still at the mercy of how well systems throw exceptions and how well are these exceptions propagated with a distributed Web Services environment.

The inherent distributed nature of Web Services and lack of source code or program binaries access makes White Box testing impossible within a SOA. With WSDLs as the de facto contract between consumers and producers in a Web Services-based SOA, significant information is available to construct intelligent and efficient Gray Box tests. WSDLs provide rich information to construct and automate such tests to improve Web Services deployments.

III. Pushing Gray towards White Box Testing

Through WSDLs – the Web Services API – testers have significant insight into the protocol, data types, operation expectations and error handling capabilities of a Web Services. Intelligent and efficient Gray Box tests can be authored and run to determine defects based on the information available in WSDLs. However, there exists a strong need to automate the test generation process based on the available information for increased testing efficiency.

XSD-Mutation™ is one such patent-pending automation technique by Crosscheck Networks' SOAPSonar™ Enterprise product. Using information available in the WSDL, a set of test cases both positive and negative can be generated to discover defects in target Web Services. The test mutations may occur at the data type, data value, message structure or protocol binding level. Although the WSDL does not reveal internal programmatic information such as relative exception handling capabilities of the two operations, through mutation generated tests; the application code exception handling logic and robustness is quickly revealed.

By using such techniques, Web Services can be thoroughly exercised without source code or binary access. Along the Black-White testing spectrum, such testing techniques push the middle ground gray Box testing more towards the White Box testing end of the spectrum without the associated expense or intrusiveness. Furthermore, given that White Box testing is not even an option in distributed Web Services-based SOAs, the only option available is to start with Gray Box testing and use automation tools such as SOAPSonar™ to push the gray towards White Box testing. Mutation techniques add newer test "frequencies" to the testing spectrum driving the test cases closer towards a complete White Box set of "frequencies."

IV. Summary & Recommendations

Web Services-based SOA plays an important role in facilitating the integration of disparate applications from various departments or trading partners and thus increasing business productivity. The distributed nature of Web Services makes Gray Box testing ideal for detecting defects within a SOA. Black Box testing provides rapid functional testing that can be used across distributed services; however, owing to the "blind" nature of Black Box testing, test coverage is limited, inefficient and redundant. White Box testing is not practical for Web Services since access to source code or binaries in a Web Services deployment is usually impossible. By leveraging the rich information presented in WSDL files, intelligent and efficient Gray Box test can be generated. Further state-of-the-art techniques such as message mutation can be used to auto-generate a large array of test that can extract program internals – exception handling, states, flows – without having access to source or binaries. Such techniques push the Gray Box testing closer to the results of White Box testing without dealing with its expense or intrusive characteristics.

V. About Crosscheck Networks SOAPSonar™

Crosscheck Networks has built SOAPSonar™ to provide you comprehensive, code-free testing that is extremely easy to set up and run. You will be generating Functional, Performance, Interoperability and Vulnerability Reports in minutes and leveraging non-intrusive and efficient Gray Box Testing Techniques through SOAPSonar™ and its patent-pending XSD-Mutation™ technology.

Contact Information

Website: www.crosschecknet.com

Email: support@crosschecknet.com

Phone: 1-888-CROSSCK (276-7725)

1 617-938-3956 (from outside US)

<Crosscheck your Web Services/>™