

Intro to SOA Regression Testing: A Hands-on Approach

Mamoon Yunus, CTO Forum Systems

Learn SOA Regression Testing techniques through automated data sources and recording base-line tests.

I. INTRODUCTION

Regress means to go backwards. Software Regression Testing is the means of identifying unintentional errors or bugs that may have been introduced as a result of changing a program module. The program module regresses by no longer working as it used to before. Software development is an iterative process in which program modules are continually modified by teams of developers to meet changing system requirements. Typical software systems with N modules have N^2 dependencies. A flaw introduced in a modified module can have significant impact across the entire system.

Regression tests help identify changes between a selected product release and a previous release of the product – called a baseline. A baseline is recorded snapshot of desirable product behavior. This expected behavior is then used to ensure that nothing has been broken in the system as a result of changes introduced in a program module. Establishing a regression testing framework is crucial for building reliable and stable software products.

Web services – the foundation of modern Service Oriented Architecture (SOA) – are self-contained, modular applications that one can describe, publish, locate, and invoke over a network. Web services are agnostic to operating system, hardware platform, communication protocol or programming language. Most IT assets such as application servers, RDBMS, CRM/ERP applications, and SaaS products now advertises their interfaces as a Web Services Definition Language (WSDL) interface ready for SOAP/XML messaging. Using SOAP for system-to-system messaging and WSDL for interface description, IT professionals now have unprecedented flexibility in integrating IT assets across corporate domains. It is this flexibility of distributed computing provided by web services that makes developing and deploying a robust, resilient and reliable Service Oriented Architecture challenging.

QA Professional faces unique challenges in performing regression testing of a Service Oriented Architecture. The fundamental advantage of a Service Oriented Architecture is reuse of services across a distributed, technology agnostic infrastructure. In as successful SOA deployment, the number and re-use of services should continue to increase. As the number of services and their re-use within a SOA increases, the difficulty in testing services increases dramatically owing to the interdependencies of the services within a distributed environment. If one the services desired behavior changes, all the dependent services will exhibit faulty behavior. Thus, SOA Architects, Developers and QA Professionals are now responsible for adapting their testing techniques, selecting appropriate testing tools and developing web services domain expertise to make their SOA deployments deliver business value reliably and securely.

In this article, we describe techniques for SOA Regression Testing through a hands-on approach that walks you through:

- Setting up a simple web services consumer (client) and producer (server) environment.
- Establishing an external MS Excel data source for driving test scenarios.
- Recording an acceptable base-line run.
- Simulate regression by changing producer service.
- Re-run external test data and identify producer service regression.

After completing the hands-on walkthrough below, QA Professionals, Developers and Architects will have a strong foundation in establishing and extending test suites for regression testing within their Service Oriented Architecture.

II. SETUP OVERVIEW

For a hands-on understanding of SOA Regression Testing, we will first build a simple web service with four operations such as *Multiply*, *Divide*, *Echo* and *Concat*. To get started, download the following components:

1. [Microsoft .NET WebMatrix](http://www.asp.net/webmatrix/download.aspx?tabindex=4): This installer includes an IDE for building web services and a lightweight web server. Download and Install WebMatrix.
<http://www.asp.net/webmatrix/download.aspx?tabindex=4>
2. [Web Service File](http://www.crosschecknet.com/SampleWS.zip): Unzip this file. It contains *SampleWS.asmx* that will be opened with .NET WebMatrix. <http://www.crosschecknet.com/SampleWS.zip>
3. [Crosscheck Networks SOAPSonar Enterprise Edition](http://www.crosschecknet.com/download/p_cust_info_try.php): Download and Install SOAPSonar, a .NET-based SOAP client used for comprehensive web services testing.
http://www.crosschecknet.com/download/p_cust_info_try.php

Components can be installed on a Windows 2000/XP/2003/Vista machine with moderate resources. The Figure below shows a typical web service deployment with a consumer-producer interaction model. The producer is the .NET WebMatrix server that supplies a web service with four operations (*Multiply*, *Divide*, *Echo*, *Concat*) that applications can invoke, typically remotely over HTTP(S). In addition, it produces the WSDL file that defines the web service interface. This file provides all the necessary information for the consumer, SOAPSonar, to send SOAP requests to the target web service. SOAPSonar consumes and interprets the WSDL-based API published by the producer and invokes the web service.

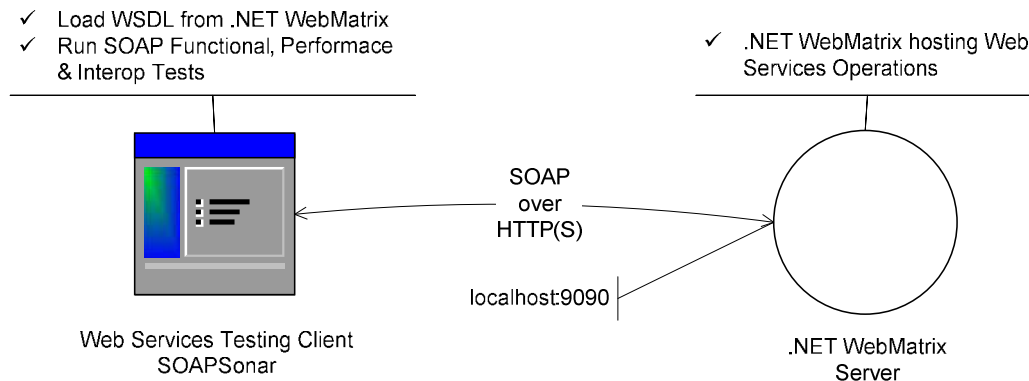


Figure 1: Web Services Consumer-Producer Setup.

III. BUILD A SIMPLE WEB SERVICE

Follow these steps to build your first web service:

1. Goto: *Start > All Programs > Microsoft ASP .NET Web Matrix > ASP .NET Web Matrix*.
2. You will be prompted with a screen. Hit Cancel.
3. Goto: *File > Open File and select SampleWS.asmx*. You will see the following C# code:

```
C:\Documents and Settings\Forum Systems\Desktop\C...
<%@ WebService language="C#" class="training" %>

using System;
using System.Web.Services;
using System.Xml.Serialization;

public class training {

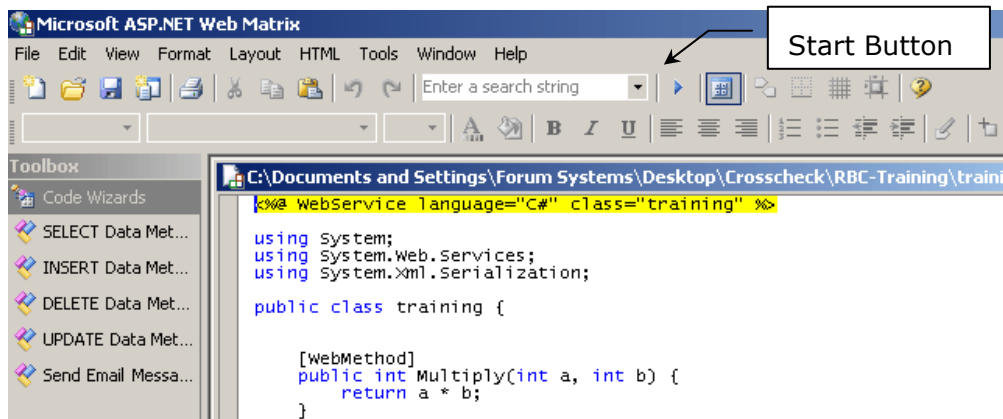
    [WebMethod]
    public int Multiply(int a, int b) {
        return a * b;
    }

    [WebMethod]
    public int Divide(int a, int b) {
        return a/b;
    }

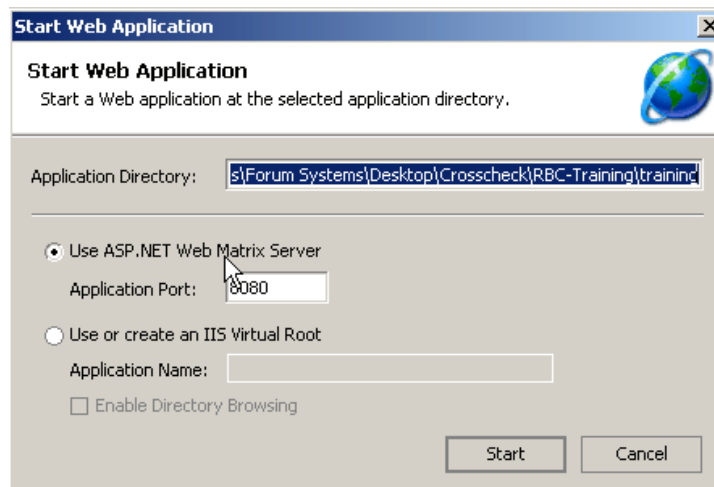
    [WebMethod]
    public string Echo(string a) {
        return a;
    }

    [WebMethod]
    public string Concat(string a, string b) {
        return a + " " + b;
    }
}
```

4. Click the *Start* button in the IDE as shown in the Figure below:





5. You will be prompted to start the web application on port 8080 (change to port 8080 if you are prompted with a different port). Make sure your local firewall is turned off.

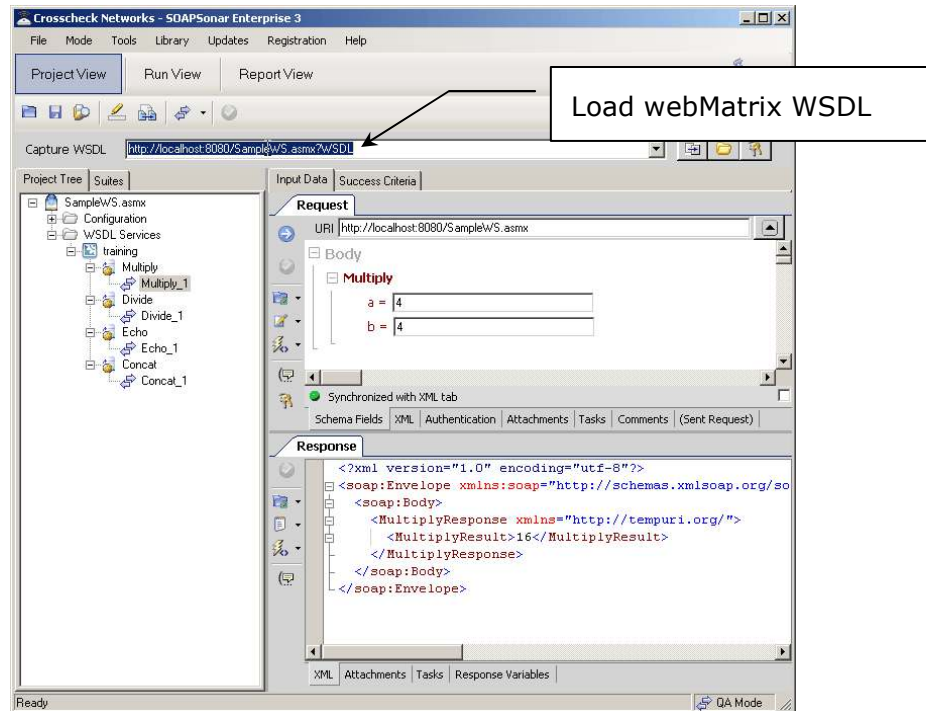


6. A web browser with a list of operation will appear. You can click the operation names and start experimenting with your first web services.

IV. SETUP TEST CLIENT

To setup the test client, perform the following steps:

1. Load the WSDL published at the .NET WebMatrix Endpoint <http://localhost:8080/SampleWS.asmx?WSDL> into SOAPSonar as shown in the figure below. Click  to commit changes and  to execute the test. You will see the SOAP response in the *Response Panel* as shown below.



2. Save the project by going to *File > Save Project As*.

With the WSDL loaded into the test client, SOAPSonar, you now have a simple consumer (SOAPSonar) to producer (webMatrix) Framework setup to perform comprehensive SOA Testing.

V. AUTOMATING DATA INPUTS

Entering input values for a SOAP message can be a manual task when a large number of input values are involved. The manual tasks can be automated by using external data sources such as RDBMS, Excel Spreadsheets or flat files that contain input values. In this Section, we will develop a test suite that uses an external Excel Data Source.

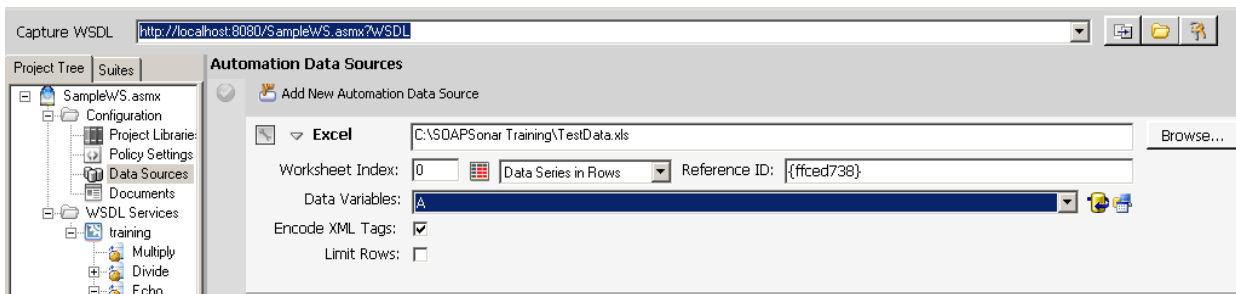


Before starting this lab, the user should ensure they switch SOAPSonar to QA mode. This can be done by selecting the Mode menu in upper left corner of SOAPSonar.

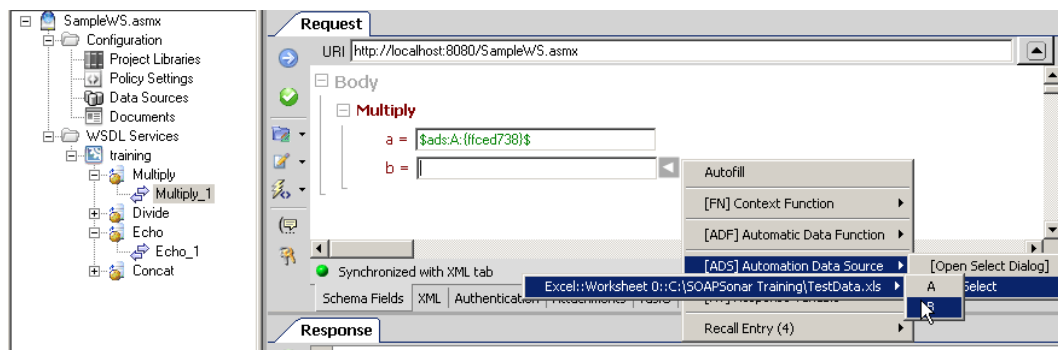
1. Create & Populate an Excel Spreadsheet *TestData.xls* with the first row labeled as *A* and *B*. Populate the values in this spread sheet. As sample Excel file is shown below.

	A	B	C
1	A	B	
2		1	1
3		2	6
4		2	5
5		2	4
6		2	3
7		10	10
8		20	12
9		56	56
10		22	57

- Under the Configuration Node in the Project Tree, select Data Sources as shown in the Figure below. Select *Add New Automation Data Source > Excel Data Source*. Browse to the *TestData.xls* and commit the changes by clicking . Verify that the Excel spreadsheet has been properly loaded, click on the icon on the screen below. The Data contents of the spreadsheet will pop-up with the *A* and *B* values from the spreadsheet.

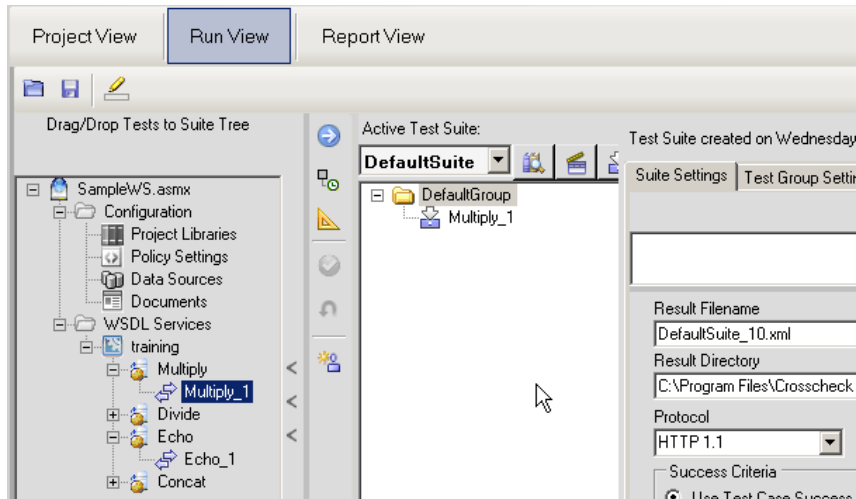


- For operation *Multiply*, populate the Test Suite *Multiply_1* variables *A* and *B* from the Automation Data Source (ADS). The ADS Dialog comes up by clicking on next to the input fields *A* and *B* as shown in the Figure below. Once you select the ADS variables as inputs of the *Multiply_1*, unique reference IDs for the data sources starting with *\$ads:* will appear each Field. Commit by clicking .



- Goto the *Run View* and Drag-and-Drop the *Multiply_1* Test Suite under the Default Group as shown in the Figure below. Commit by clicking . Run test cases by clicking . Once the test suite is executed, click on View Detailed Log Results. With nine *A*, *B* pairs in our spreadsheet, the test case

iterates through all nine value and multiplies the inputs as expected.



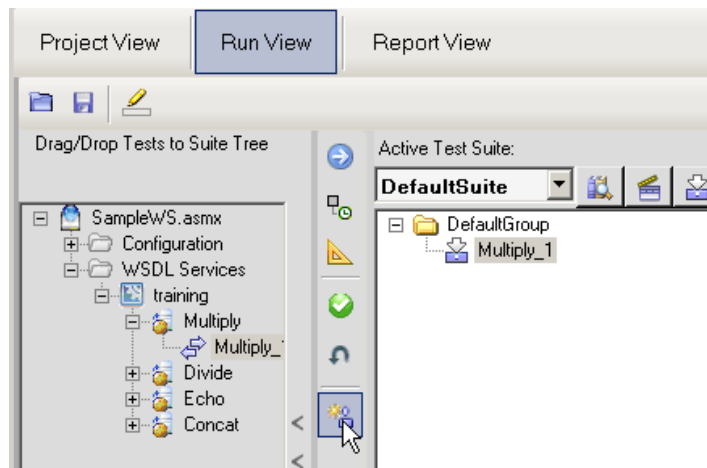
5. Remove a row from the spreadsheet *TestData.xls*. Re-run the Test Case. The test suite now runs eight times.

Testers can generate comprehensive test suite values by simply pointing to a data source such as an Excel spreadsheet or a RDBMS. This eliminates the need to manually enter values for test cases. You should now be comfortable with automating your test suites using external data sources.

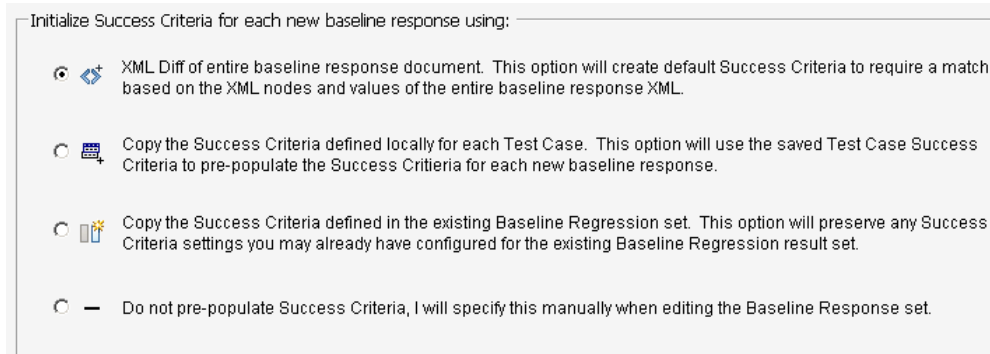
VI. SOA REGRESSION TESTING

Once a SOA tester has built an automated test suite, the next step is to determine whether the target web services operation is behaving as expected. This is followed by recording the Baseline Response set for the operations and then periodically running the test suite to ensure that the web service operation has not regressed. In this Section, we will record base line regression responses, modify the web service code and then re-run the test suite to highlight target web service regression issues.

1. Goto *Run View* and click on *Generate New Regression Baseline Response Set* as shown in the Figure below.



- Choose the Default value XML Diff of Entire Document as the Baseline success criteria as shown in the Figure below. This will run the Excel Data Source and record the response values as the base line. Review the baseline values in the *Test Suite Regression Baseline Editor* and press *OK* to continue.



- In Microsoft ASP.NET Web Matrix, change the Multiply Operation to a new operation such as $a * b * 2$. This simulates an operation change that should now be detected by the recorded regression response set. The operation change is shown below. Make sure to save the file after changing the operation.

```

C:\Documents and Settings\Forum Systems\Desktop\Crossch
<%%@ WebService language="C#" class="training" %>

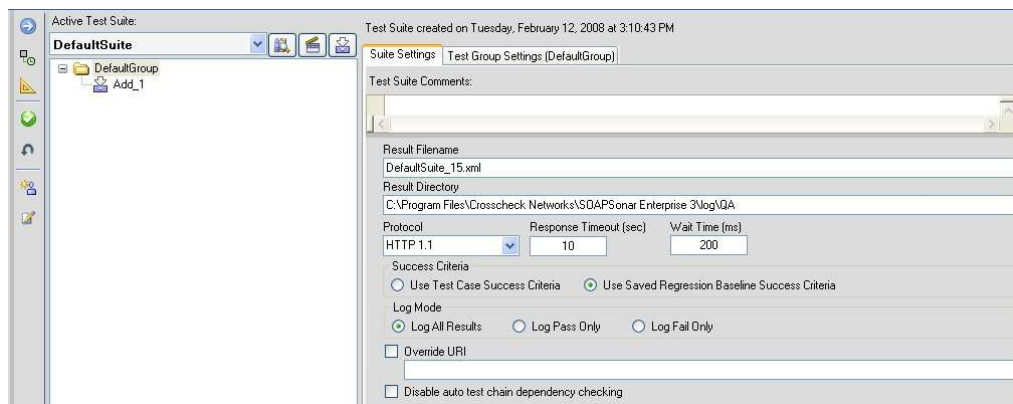
using System;
using System.Web.Services;
using System.Xml.Serialization;


public class training {

    [WebMethod]
    public int Multiply(int a, int b) {
        return a * b * 2;
    }
}

```

- In SOAPSonar, under Run View, change the Suite Setting select the Use Saved Regression Baseline Success Criteria radio button as shown in the Figure below.



- Re-run the test suite by clicking on . With the new operation Multiply with $a * b * 2$ as the operation instead of the original operation $a * b$, all the responses fail since the return values are twice that of the base line Multiply return values.

Click on *View Detailed Log Results* on the Real Time Run Monitor. All Tests show Test Results as *Failed* indicating that a systemic regression issue has been introduced for the target web service. The web service operation *Multiply* has therefore regressed.

VII. CONCLUSIONS

The promise of web service-based SOA lies in re-usability across distributed environments. The ease of developing web services and interdependencies between services puts a significant burden on SOA Testers to ensure that web services are robust, reliable, secure and scalable. Disciplined Regression Testing is paramount in ensuring that web services behave as expected throughout the SOA Lifecycle. Through collaboration, a growing understanding of web services technologies, and comprehensive SOA Regression framework, sophisticated SOA Testing Tools, SOA team can ensure that high quality services are deployed within an enterprise.

VIII. ABOUT CROSSCHECK NETWORKS

Crosscheck Networks is focused on providing products for **SOA Testing**. Crosscheck's flagship product, **SOAPSonar**, reduces the pain point of testing Web Services. It provides comprehensive code-free web services testing with an intuitive interface. Simple navigation, Drag/Drop WSDL loading, Drag/Drop Test Suite Management, Complex WSDL Support (WSDL and XSD imports) and sophisticated Performance techniques make it easy to validate your web services. SOAPSonar provides comprehensive testing across the Four Pillars of SOA Testing: Functional Regression, Performance Testing, Interoperability Conformance and Vulnerability Assessment.