Testing of Web Services using Behavior-Driven Development

Ahmet Furkan Oruç and Tolga Ovatman
Department of Computer Engineering, Istanbul Technical University, 34469, Maslak, Istanbul, Turkey

Keywords: Behavior-Driven Development, Gherkin, JMeter, Software Testing, Testing of Web Services, Web Services.

Abstract: Web services are commonly used in the communication of software over the web. To fully trust a web service, it should be tested and certified, but testing of web services provoke new challenges. Behavior-Driven Development (BDD) can be applied to the testing of web services. Gherkin language is used to define scenarios in BDD. We used Gherkin language to define test cases for web services and we developed a tool to convert these test cases into JMeter test scripts.

1 INTRODUCTION

A web service is defined as ”a software system designed to support interoperable machine-to-machine interaction over a network” by W3C (World Wide Web Consortium, 2015). Web services provide a pre-defined web platform, where different applications can interoperate with each other. Web services are commonly used in exchanging data over the internet.

One of the biggest issues about web services is the trust issue. A web service should work correctly every time it is executed. This is only achieved via tests and certification. Since testing of web services only from client side is not enough to issue the trust, different solutions should be applied. To test a web service, following functionalities should be tested: Basic web service functionality, web service interoperability, SOA functionalities, quality of service and load/stress testing (Tsai et. al., 2005).

Behavior Driven Development (BDD) is a software development process, in which the requirements and the expected behavior of the system are specified in a human readable, ubiquitous language to be able to perform acceptance tests (Lopez-Pellicer et. al., 2014). According to Evans (2003), the ubiquitous language is defined as a common language between developers and domain experts to understand business specifications. With the use of such a language, behavior of the system is specified before development and unit test cycles. An example to such a language in today’s software industry is called Gherkin. Gherkin is a business readable, domain specific language created specifically for behavior descriptions (Behat, 2016). Typical Gherkin syntax can be seen in Figure 1.

```
Feature: Some terse yet descriptive
text of what is desired
In order to realize a named business
value
As an explicit system actor
I want to gain some beneficial outcome
which furthers the goal
     Additional text...

Scenario: Some determinable business
situation
Given some precondition
And some other precondition
When some action by the actor
And some other action
And yet another action
Then some testable outcome is achieved
And something else we can check happens
too

Scenario: A different situation
```

Figure 1: A sample Gherkin document.

Conformance testing is the process to ensure if a system fulfills the requirements (Gray et. al., 2010). In addition, web service performance testing is the process to determine if a web service replies within a limited time in an environment where many users access the web service concurrently. There are some available tools for testing web services like SoapUI (SmartBear, 2016) and JMeter (Apache Software Foundation, 2016). These tools provide a GUI to create test plans, run the tests and view the results. Variety of these tools and other web service testing

Oruç, A. and Ovatman, T.
Testing of Web Services using Behavior-Driven Development.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 2*, pages 85-92
ISBN: 978-989-758-182-3

approaches are inspected in a survey paper which is written by Bozkurt et. al. (2013).

There is another paper about Behavior-Driven Development web service testing approach which is written by Lopez-Pellicer et. al. (2014), which is applied to the conformance testing of INSPIRE web services. However, this work mostly focuses on INSPIRE web services.

In a survey which inspects the publications written in behavioral software engineering field, it is mentioned that there are only two publications regarding software testing (Lenberg et. al., 2015). This result proves how software testing field is open for more studies. In this work, we use the power of the Gherkin language to define test cases for web services. These test cases may include either conformance or performance tests for the given web service. Then, we develop a tool to convert the Gherkin language into a test script which can be run by using one of the most widely used testing tools which is JMeter (Apache Software Foundation, 2016). JMeter test scripts are easily modified and may contain different types of tests and scenarios. As a result, creating and doing the actual test and observing the results can be done by any domain expert who does not required to have software knowledge.

In section II, we have explained basic software testing, TDD and BDD concepts, in section III, we presented how we applied BDD to the conformance and performance testing of web services, in section IV, we gave a simple application on how to use the tool we developed and shared a sample test and results of the test and finally in section V, we discussed the results of our work.

2 APPLYING BDD APPROACH TO THE SOFTWARE DEVELOPMENT CYCLE

According to Sommerville (2007), software testing process has two distinct goals. First, to ensure that the software meets its requirements. Second, to discover parts in the software where the behavior of the system is wrong. In Figure 2, a standard model for the software testing process is given.

Test Driven Development (TDD) is a software development cycle which is defined as writing the unit tests before writing the production code (Osherove, 2009). On the other hand, traditional method for software development emphasizes the opposite which is testing after development.

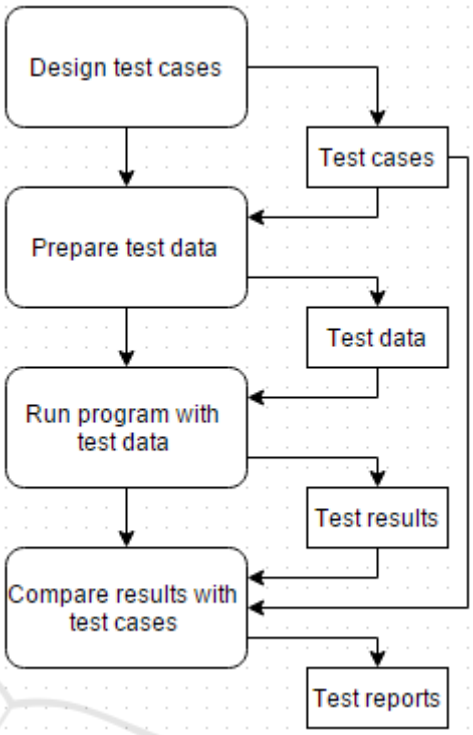

Figure 2: A model of the software testing process (Sommerville, 2007).

According to Osherove (2009), the traditional way of writing unit tests is given in Figure 3, while TDD cycle is given in Figure 4.

Behavior Driven Development (BDD) has emerged from Test Driven Development (TDD). In BDD, the requirements and the expected behavior of the system are specified in a human readable, ubiquitous language to be able to perform acceptance tests (Lopez-Pellicer et. al., 2014).

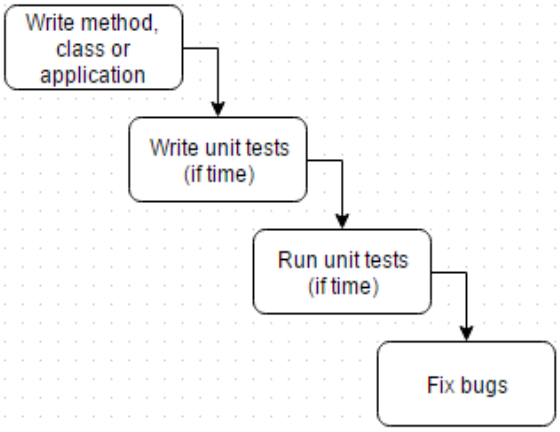

Figure 3: The traditional way of writing unit tests (Osherove, 2009).

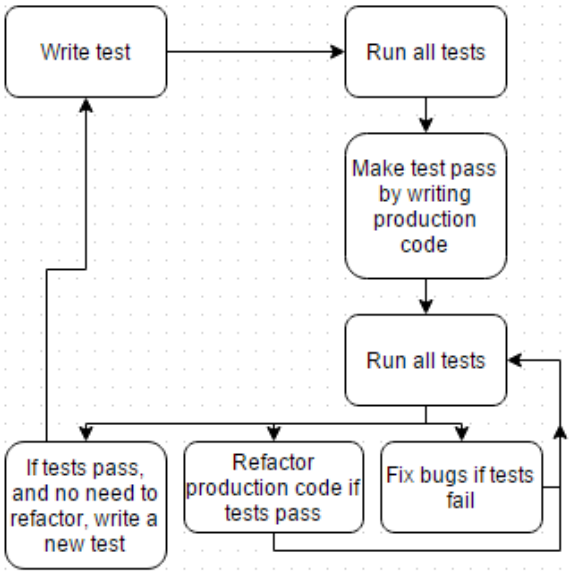

Figure 4: Test Driven Development - A bird's eye view (Osherove, 2009).

During acceptance tests, the system requirements, the tests and the actual code may be re-factored. As a result, domain experts are joined to the software development cycle and software is written according to the tests while tests are written according to the system requirements.

3 CONFORMANCE AND PERFORMANCE TESTING OF WEB SERVICES USING BDD

Conformance and performance testing of web services are achieved by the use of testing tools. One of the most accepted unit testing tools used in software industry is JMeter (Apache Software Foundation, 2016). With using JMeter, many unit test cases for web services can be completed. To achieve desired tests, a test plan should be created from JMeter GUI. Each test plan is saved as a JMeter script (xml file). These test plans may contain multiple threads, http requests, response assertions, response graphs, shell scripts, database scripts etc. Test plans can run either from GUI or using a terminal command. Once the test plan is run, results can be viewed via listener components added to the plan.

In our work, we used Gherkin language to create JMeter test plans dynamically. By this, we achieved two benefits. First; It removes the need of developing unit testing software, since we are using JMeter to do the tests. Second; since Gherkin is a business readable, domain specific language, unit tests can be created and run by domain experts, instead of developers. In addition to these benefits, this method removes the possibility of bugs which may occur during testing software development.

Table 1: Gherkin string variable explanations.

Part	Variable	Explanation
Scenario	Operation Name	Name of the operation which is stated in the configuration file
Given	Some, a	Number of threads to be used in test (Some: stated in configuration file)
Given	With/out	With: Existing users are used, Without: Random users are used (Both has different jdbc queries which are stated in the configuration file)
Then/And	Positive/ Negative	Used for response assertion. Real response values are stated in the configuration file.
Then/And	Duration	Used for duration assertion.

Generation of JMeter test plans are based on a Gherkin string and a configuration file. We developed a Gherkin parser tool which requires a proper Gherkin string and a configuration file as inputs and generates JMeter test plan as the output. Configuration file is a simple xml file which contains the technical information about related web service methods. This information includes; operation name, web service url, web service action, request xml, number of threads to test, response assertion key and values, parameters to be used in request, jdbc configuration and jdbc queries. So, the configuration file contains the technical information about the web services to test, while the Gherkin string contains the test case scenarios written in a domain specific language.

The Gherkin language format we support with the tool is given in Figure 5. In the Gherkin string, the parts which are shown with braces are considered as variables by the Gherkin parser tool. During the generation of the JMeter test plan, variables are mapped to the values which are taken

```
Scenario: Scenario explanation
{Operation Name}

Given {some, a} user{s} {with/out}
something

When the user asks for present status

Then a {positive, negative} answer
should return

And response time should be lower than
{duration} miliseconds
```

Figure 5: A sample Gherkin document.

from the configuration file. The variables used in Gherkin string are explained in Table I. Configuration file contents are also explained in Table II.

The tool we developed, parses the Gherkin string and uses the configuration file to create the JMeter script. It also has the option to run the JMeter script with a command, print to results to screen and write results to a jtl file. The tool GUI can be seen in Figure 6.

The functionalities of the tool can be inspected in 3 steps. First step is to generate sample configuration file. The tool generates a sample configuration file, which should be modified with the technical information of the web service to test. Second step is to generate JMeter script. In second step, tool asks for a configuration file (which is generated in Step 1), and by using the Gherkin stated in GUI, it generates the JMeter script. After that the script can be opened using either JMeter GUI or can be tested directly from our tool which is the third step. After running the test, results are printed to the GUI and written to a jtl file for further inspection. JMeter GUI is also given in Figure 7.

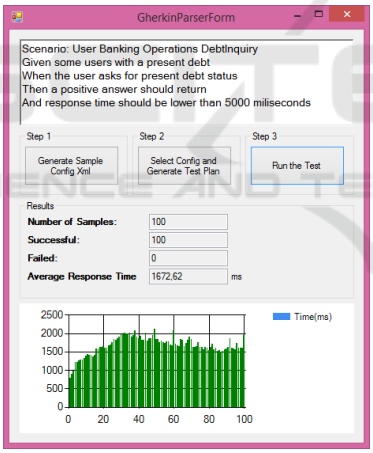

Figure 6: Testing tool GUI.

Table 2: Contents of the configuration file.

Node	Parent Node	Explanation
TestConfig	-	Root node
MethodList	TestConfig	List of web service methods
Method	MethodList	Contains information regarding the method
Name	Method	Name of the web service method
Url	Method	Url of the web service
Action	Method	Web service action
Request	Method	Web service request body which may also contain parameters
ThreadCount	Method	Number of threads to be used in test
AssertionList	Method	List of the assertions
Assertion	AssertionList	Created for each assertion
Key	Assertion	Key value used in Gherkin
Object Name	Assertion	Object to do the assertion
ObjectValue	Assertion	Value of the object to assert
ParameterList	Method	List of the parameters
String	ParameterList	Name of the parameter
JDBCConfig	Method	Main JDBC configuration node
Url	JDBCConfig	JDBC connection string
Driver	JDBCConfig	JDBC driver
Username	JDBCConfig	JDBC username
Password	JDBCConfig	JDBC password
JDBCQueryList	Method	List of the JDBC queries
JDBCQuery	JDBCQueryList	Created for each JDBC query
Key	JDBCQuery	Either Random/Existing
Value	JDBCQuery	Query to get either random/existing record form the database

Table 3: Invoice WCF service information.

Method Name	Request	Response	Explanation
Debt Inquiry	**SubscriptionNo:** User subscription number	**InvoiceInfo:** Object containing the invoice information **IsDebtFound:** Flag assigned if any debt found for user **Result:** Object to show if the operation is successful	Users inquire their debts using this method.
Collection	**SubscriptionNo:** User subscription number	**IsPaid:** Flag assigned if the payment is successful **Result:** Object to show if the operation is successful	Users pay their bills using this method.
Collection Cancel	**SubscriptionNo:** User subscription number	**IsCancelled:** Flag assigned if the payment is cancelled successully **Result:** Object to show if the operation is successful	Users cancel their payments using this method.

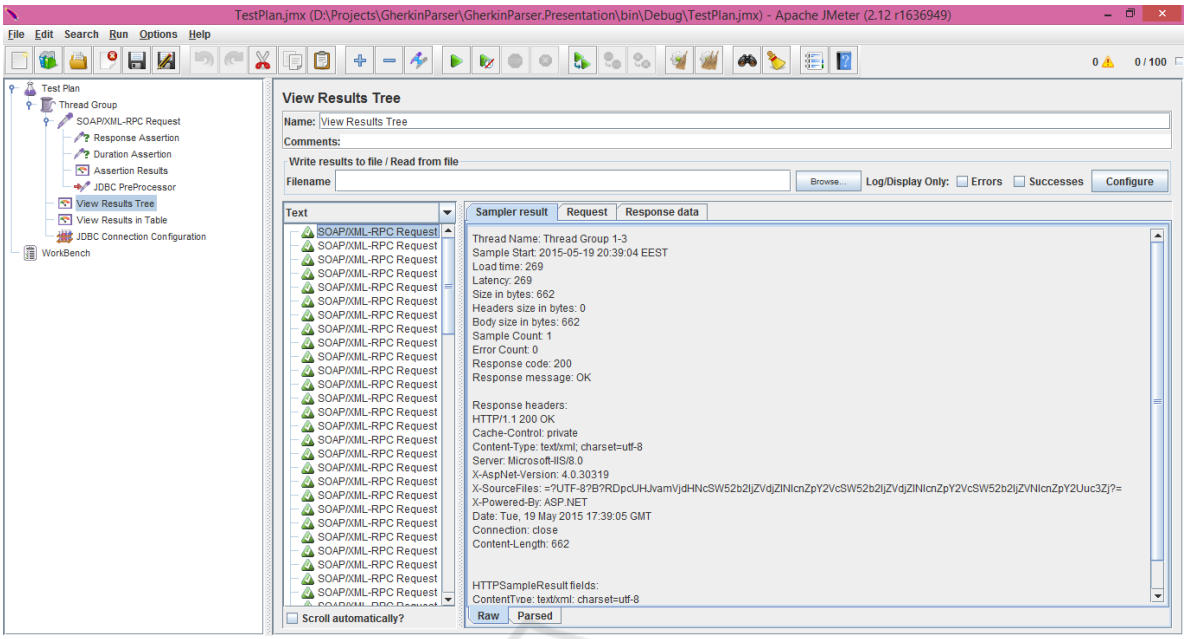

Figure 7: JMeter GUI.

The Gherkin used in this application can be seen in Figure 8. "Scenario" defines the name of the operation in the web service to test. Here, "DebtInquiry" is given as the operation name.

In the "Given" part, two things are taken into consideration. The keyword "some" indicates that multiple threads will be used in testing. Number of threads used is set in the configuration file. The keywork "with" indicates that the existing users are used in the test. To obtain an existing user, JBDC queries are used to select a random user from the database. JDBC configuration and the query are set in the confiugration file.

In the "Then" part, "positive" keyword is used to point out that the response of the web service should be positive for the case. Any logical response assertion can be used in this part to achieve conformance testing.

```
Scenario: User Banking Operations
DebtInquiry
Given some users with a present debt
When the user asks for present debt
status
Then a positive answer should return
And response time should be lower than
5000 miliseconds
```

Figure 8: Gherkin file used in application.

In the "And" part, response time assertion is achieved. Responses which takes more than 5000 miliseconds is marked as failed. This fulfills the performance testing issue.

The configuration file used in the test is given in Appendix A.

After writing down the Gherkin string and creating the required configuration file, JMeter test script is generated using the tool. Then, actual test can be completed in two different ways. First, the test can be run from the tool. This method runs a command which runs the JMeter test script. Second method is to open the created JMeter script with the JMeter GUI and run the test from the GUI. Either way, test results are written to jtl file on the test script location. By using JMeter GUI, sample test request can be written as in Figure 9; sample test response can be seen in Figure 10. The results of the JMeter test is given in Figure 11.

4 RESULTS AND EVALUATION

Test results contain multiple outcomes according to the given scenario. The results may answer the following questions:

1. Is the web service running? Is it responding to the requests?
2. Is the web service responding within the expected period of time? (Performance)
3. Is the response expected? (Conformance)

By the use of our tool, it is easy to re-run all performance and conformance tests with different parameters and values. Therefore, it reduces all the

time and effort required to create different tests. In addition, evaluation of the test results are also easy with the GUI we propose. Also, the same GUI is used to create different performance and conformance tests using Gherkin language. So, creation, running and evaluation of the test results can all be accessed from the same GUI and it can be used by all domain experts. We believe this is very useful in terms of time and effort.

```
<soapenv:Envelope
<!-- XML Schemas -->>
  <soapenv:Header/>
  <soapenv:Body>
    <tem:DebtInquiry>
      <tem:request>
       <inv:SubscriptionNo>
           222222222
        </inv:SubscriptionNo>
      </tem:request>
    </tem:DebtInquiry>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 9: JMeter request script.

```
<s:Envelope
 <!-- XML Schemas -->>
  <s:Body>
    <DebtInquiryResponse
      xmlns="http://tempuri.org/">
      <DebtInquiryResult
      <!-- XML Schemas -->>
        <a:InvoiceInfo>
          <a:InvoiceAmount>
           35.25
           </a:InvoiceAmount>
          <a:InvoiceDueDate>
             2015-05-15T00:00:00
           </a:InvoiceDueDate>
          <a:InvoiceNo>
         3
        </a:InvoiceNo>

           <a:NameSurname>
              Ali
            </a:NameSurname>
        </a:InvoiceInfo>
        <a:IsDebtFound>
         True
        </a:IsDebtFound>
        <a:Result>
         <a:ErrorMessage i:nil="true"/>
           <a:IsSuccessful>
              True
         </a:IsSuccessful>
        </a:Result>
      </DebtInquiryResult>
    </DebtInquiryResponse>
  </s:Body>
</s:Envelope>
```

Figure 10: JMeter response code.

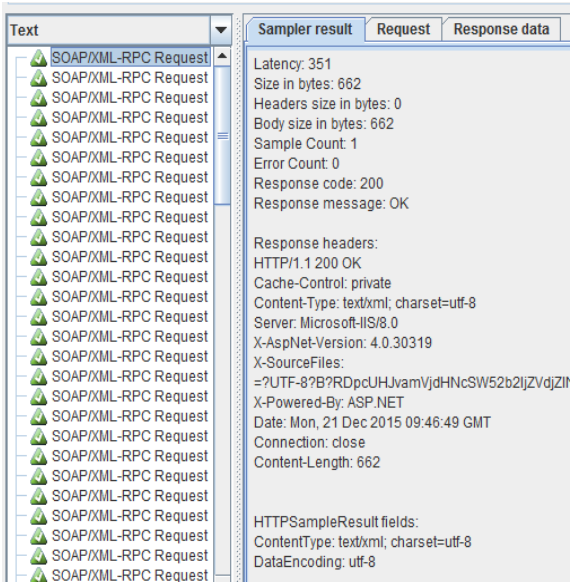

Figure 11: JMeter test result.

Table 4: JMeter unit time requirements.

JMeter Component	Unit Time Required	Explanation
Thread Group	1	Used for creating multiple thread requests
Soap/ Xml-Rpc Request	10	Used for preparing web service request body, parameters, action and url
Response Assertion	3	Used for conformance testing
Duration Assertion	1	Used for performance testing
Assertion Results	1	Used for monitoring assertion results
JDBC PreProcessor	3	Used for generating existing parameters from a database
View Results Tree	1	Used for monitoring results
JDBC Connection Configuration	5	Used for configuring database connection

For better understanding of the benefits of the tool we propose, we have prepared a unit time cost table which is given in Table 4. This table contains a symbolic units of work required to create each component using JMeter. Values given in the table are only our estimation. By using this table, the time required to create a test plan using JMeter and using our tool can be compared.

According to Table 4, 25 units of time is required to create a single web service test plan in JMeter. This time might be reduced for similar tests, but still it should be prepared by some technical expert. With the tool we propose, this time is greatly

reduced and any domain expert may prepare and apply new acceptance tests. In Table 5, unit time requirements for the tool we propose is given.

As a result, it takes 12 units of time to create the first test plan. After that, sequential test plans only require 2 units of time. In addition, they can be prepared by any domain expert.

Table 5: Web service testing tool unit time requirements.

Operation	Unit Time Required	Explanation
Preparation of configuration file	10	The technical information about the web service is prepared by the developer
Preparation of Gherkin string	2	Any domain expert may prepare the tests using Gherkin

5 CONCLUSIONS

We have presented a new tool to ease the creation of web service test scenarios and test software. The tool we presented, uses the Behavior-Driven Development concept with Gherkin language to generate test scripts dynamically. First benefit of our tool is: since Gherkin is a domain specific language, any domain expert without software knowledge can create and run web service tests. Second, developers are not required to write unit tests manually, since we used the powerful testing tool JMeter. We believe that similar BDD testing approaches will become more popular in the near future.

REFERENCES

Apache Software Foundation, 2016. *Apache JMeter*. Available at: http://JMeter.apache.org/ (Accessed at: 10 January 2016).

Behat, 2016. *Writing Features*. Available at: http://docs.behat.org/en/latest/guides/1.Gherkin.html (Accessed at: 10 January 2016).

Bozkurt M., Harman M., Hassoun Y., 2013. Testing and verification in service-oriented architecture: a survey. In *Softw Test Verif Reliab* vol. 23 pp. 261–313.

Evans E., 2003. *Domain-driven design*, Addison-Wesley Professional, Boston, pp. 123–135.

F. J. Lopez-Pellicer, M. Á. Latre, J. Nogueras-Iso, F. J. Zarazaga-Soria, J. Barrera, 2014. Behaviour-Driven Development Applied to the Conformance Testing of INSPIRE Web Services. In *Connecting a Digital Europe Through Location and Place Lecture Notes in Geoinformation and Cartography,* pp. 325-339.

Gray M., Goldfine A., Rosenthal L., Carnahan L., 2010. Conformance testing. In *Information technology laboratory, NIST*. Available at: http://www.nist.gov/itl/ssd/is/conformancetesting.cfm (Accessed at: 10 January 2016).

Lenberg, Per, Robert Feldt, Lars Göran Wallgren, 2015. Behavioral Software Engineering: A Definition and Systematic Literature Review. In *The Journal of Systems and Software* vol. 107 pp. 15-37.

Microsoft Developer Network, 2016. *What Is Windows Communication Foundation*. Available at: https://msdn.microsoft.com/en-us/library/ms731082%28v=vs.110%29.aspx (Accessed at: 10 January 2016).

SmartBear, 2016. *SoapUI*. Available at: http://www.soapui.org/ (Accessed at: 10 January 2016).

Sommerville, Ian, 2007. *Software Engineering*. Addison-Wesley. Harlow, England, 8th ed., pp. 537-540.

Osherove, Roy, 2009. *The Art of Unit Testing: With Examples in .NET*. Manning, Greenwich, pp. 42-44.

W. T. Tsai, X. Wei, Y. Chen, and R. Paul, 2005. A robust testing framework for verifying web services by completeness and consistency analysis. In *SOSE '05: Proceedings of the IEEE International Workshop*, pp. 151–158.

World Wide Web Consortium, 2015. *Web Services Glossary*. Available at: http://www.w3.org/TR/ws-gloss/ (Accessed at: 10 January 2016).

APPENDIX A

```
<TestConfig
<!-- XML Schemas -->>

  <MethodList>
    <Method>
     <Name>DebtInquiry</Name>
    <Url>
          http://localhost:4444/Invoic
     eService.svc?wsdl
     </Url>
    <Action>
          http://tempuri.org/IInvoiceS
     ervice/DebtInquiry
     </Action>
     <Request>
    <!--SOAP Request -->
    </Request>
      <ThreadCount>100</ThreadCount>
      <AssertionList>
        <Assertion>
          <Key>Positive</Key>
          <ObjectName>
            IsDebtFound
             </ObjectName>

          <ObjectValue>
            true
          </ObjectValue>
        </Assertion>
```

```
        <Assertion>
          <Key>Negative</Key>
          <ObjectName>
            IsDebtFound
          </ObjectName>
          <ObjectValue>
            false
          </ObjectValue>
        </Assertion>
      </AssertionList>
      <ParameterList>
        <string>SubscriptionNo</string>
      </ParameterList>
      <JDBCConfiguration>
       <Url>
       <!--DB URL -->
       <Url>
<Driver>
       <!—DB Driver -->
       </Driver>
       <Username />
       <Password />
      </JDBCConfiguration>
      <JDBCQueryList>
        <JDBCQuery>
          <Key>Existing</Key>
          <Value>
             SELECT
                 TOP 1 SubscriptionNo
                 FROM Invoice
                 WHERE IsPaid = 0
                 ORDER BY NEWID()
           </Value>
        </JDBCQuery>
        <JDBCQuery>
          <Key>Random</Key>
          <Value>SELECT RAND()</Value>
        </JDBCQuery>
      </JDBCQueryList>
    </Method>
    <Method>
        <!--Other web service
           methods...-->
   </Method>
  </MethodList>
</TestConfig>
```

Figure 12: Configuration file used in the test.