UNIVERSITY OF WATERLOO
Software Engineering

Analysis of Web Application Functional Testing Options

Telus Health Inc.
Mississauga, ON L4W 4T9

Prepared by

Daiwei Fan
Student ID: 20458752
User ID: d3fan
2A Software Engineering
April 11, 2014

Daiwei Fan
#48, 461 Beechwood Place
Waterloo, ON
N2T 2N8

April 11, 2014

Andrew Morton, Director
Software Engineering Program
University of Waterloo
200 University Ave. West
Waterloo, Ontario. N2L 3G1

Dear Professor Morton:

The enclosed report, *Analysis of Web Application Functional Testing Options*, is the first work report that I have written. The information presented in the report is based upon my recently completed 2A work term with Telus Health Inc..

Telus Health provides health and related transaction record for clients of medical insurance companies. I was situated in the new VIP room team, which is responsible for rebuilding a web service for clients to look up their drug claim transaction history. During my time in Telus, I was assigned to study the pre-existing interface and to extract data from existing log files.

The following report performs an analysis on the different techniques for testing features implemented in VIP room. This problem is one that I encountered during my work term. My team had to decide which method was best suited to the task given all the requirements and our abilities.

I would like to thank my manager Kevin Ho and Quality Analyst Anshuman Ghandi, both of whom gave inputs on how functionality testing is done in other projects produced by Telus Health. I hereby confirm that I have received no help, other than what is mentioned above, in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Daiwei Fan
Student ID: 20458752

Encl.

Executive Summary

Telus Inc., a leading Electronic Medical Record (EMR) provider in Canada, is designing VIP room, a drug claim searching web application for patients and insurers. A major step in the designing phase involves a decision of choosing an approach/method to test the application. This report provides a comparison of three alternatives, UI testing, SOAP testing and JUnit testing, assessing their suitability to the application.

This report analyses the three approaches/methods against four criteria: Time, Automation, Reproduction and Productivity. A decision matrix is used to determine the most suitable approach/method.

The three methods/approaches excel in different criteria. UI testing performs well in Time, SOAP leads in Reproduction and JUnit holds the standard for Productivity. UI testing is not successful in Automation while SOAP and JUnit performs equally well in the most important aspect of functional testing.

Ultimately, the decision matrix indicates that JUnit is the most suitable solution. However, the close score from SOAP affected the decision making. It is finally determined that both JUnit and SOAP will be used to test VIP room to compensate the shortcoming of both methods/approaches. This report recommends that the SOAP test cases to be carefully labelled and grouped in order for quality analysts to share and JUnit test case modifications to be severely managed by version control to avoid mass build failures.

Table of Contents

List of Figures

List of Tables

1 Introduction

Telus Health is a "TELUS Health is a leader in telehomecare, electronic medical and health records, consumer health, benefits management and pharmacy management."[1]. With the recent acquisition of Med Access Inc., Telus is "confirmed as the largest EMR (Electronic Medical Record) provider in Canada"[1]. This title has encouraged Telus to provide service with higher quality. In January 2014, I was assigned to the VIP room to implement a web service that mainly contains a drug claim searching and benefit management feature. This particular feature allows different levels of users such as patient, insurer and pharmacist to access the patient's record of drug claims and payments. After a certain amount of designing and coding, it came to the point where some basic testing for the code was necessary. I believe that even elementary testings should be carefully designed for future documentation and reproduction.

Functionality testing has always been one of the most important tasks in the quality assurance (abbreviated as QA) process. All companies are working to discover a testing method which is able to cover all sources of error, requires minimal human effort and consumes least amount of time. It is especially important for Telus Health, a company which has adapted the Agile software development system, to have a highly efficient QA process in order to accommodate the fast pace of the system. At Telus, agile development is structured as series of "sprint cycles." Every aspect of the development process, including technical requirements, design etc., are continually revisited throughout the lifecycle of the project. As a consequence, a series of developing and testing will take place. As illustrated in Figure 1, the testing phase appears on a intense frequency, which results in many high demands on the tests' quality, efficiency etc. These demands will be considered as criteria to evaluate a testing approach later on in the report.

This report provides an evaluation of three functionality testing methods, assessing their suitability to the VIP room web service. You will first be introduced to the VIP room work flow. Then, the three approaches for testing will be explained and compared in detail. Comparisons

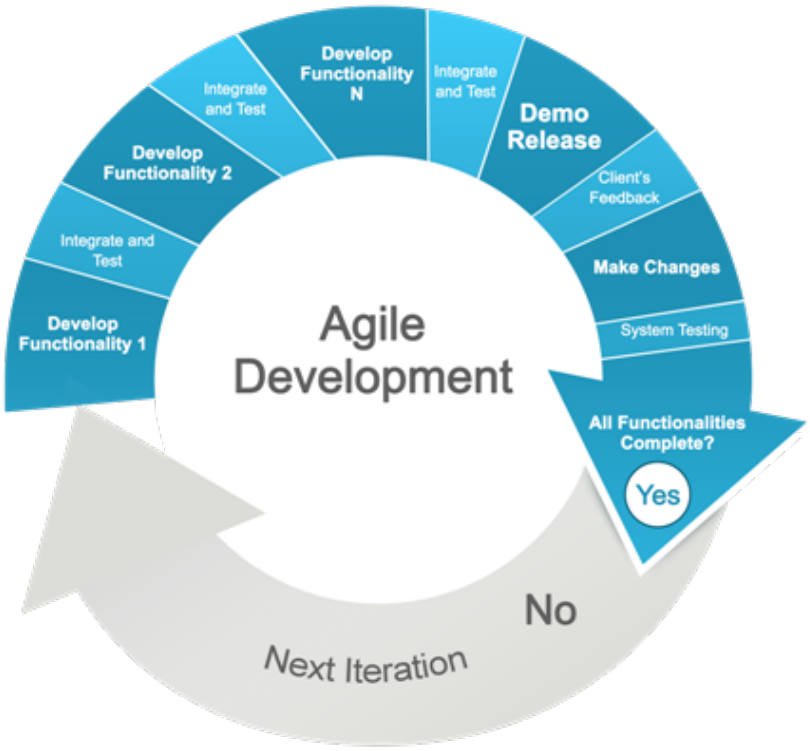

Figure 1: Agile development workflow

will be made among various criteria and will be finalized using the Decision Matrix Analysis. Finally, the report summarizes the results of the analysis, draws conclusions, and makes a recommendation. The reader is expected to have reasonable background in web service development. Basic knowledge in the Extensible Markup Language (XML)would also help the reader quickly understand one of the approaches. Other technology and terminologies used in this report will be explained on their first appearance.

2 Problem Specification

2.1 Background

The web application uses Apache Maven as its build manager and Apache Tomcat as server. It utilizes the traditional three-tier architecture. Figure 2 depicts the sequence diagram of a three-tier web service workflow:

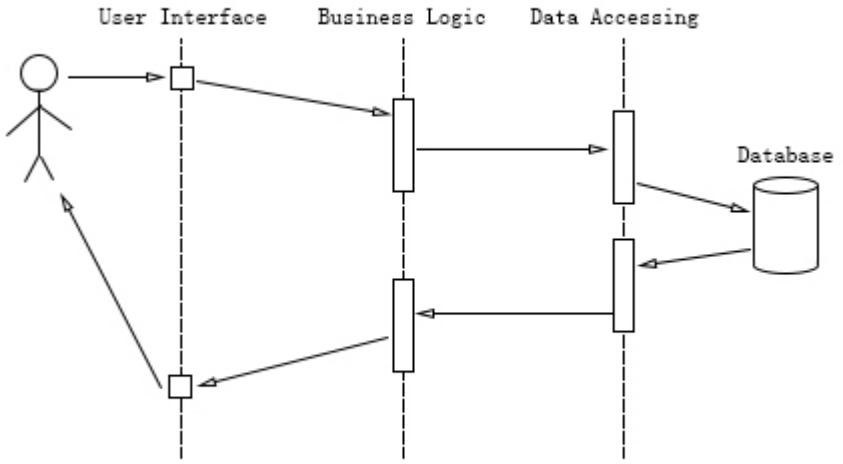

Figure 2: three-tier web service sequence diagram

2.2 Options

The three testing approaches sends data to different part of the flow.

2.2.1 Manual testing from User Interface

The most naive method is to manually enter input data on the User Interface (UI)

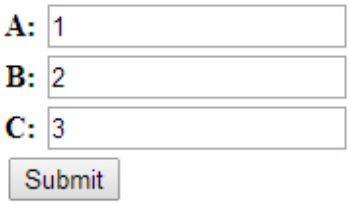

Figure 3: input from simple UI

2.2.2 Simple Object Accessing Protocol (SOAP)

This method uses the Simple Object Accessing Protocol. "SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment."[2] It wraps the input data in an envelope and sends it as a request directly to the business layer. A soap request template is populated based on a file written in Web Services Description Language (WSDL). This file can be generated from a Java Class containing business logic. In most cases this Class is the controller or entrance to the business layer. The following is a screen shot of a SOAP client called "SoapUI":

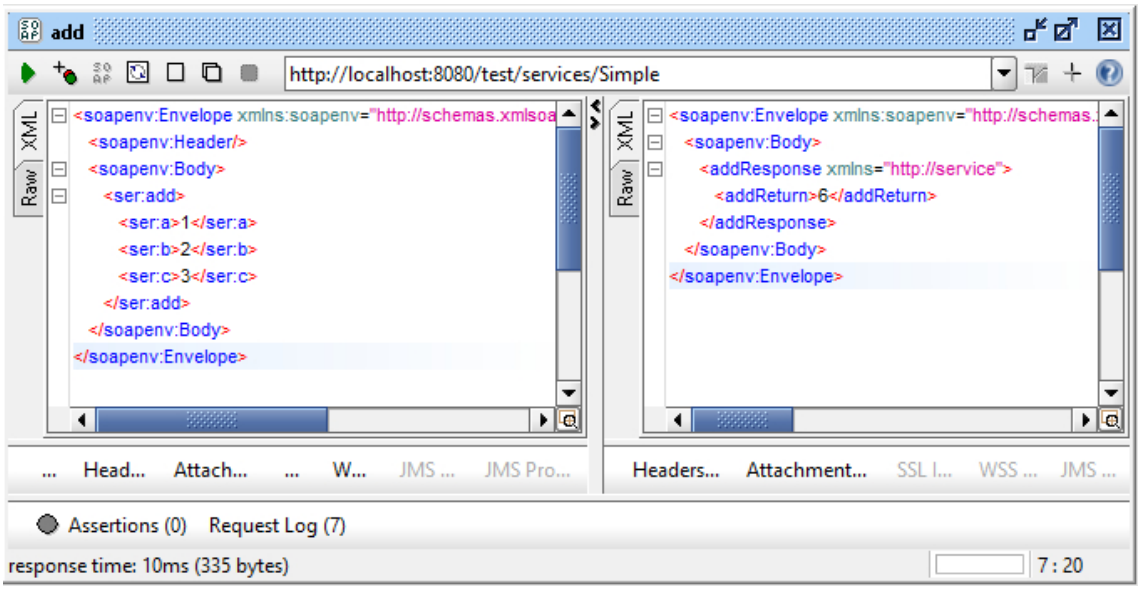

Figure 4: SoapUI request/response

The XML message on the left is a simple request containing three parameters and the one on the right is its response. This request template is populated from WSDL file (shown in appendix)

generated by the following Java Class:

```
// Simple.java
public class Simple {
	public int add(int a, int b, int c){
		return a + b + c;
	}
}
```

In short, SOAP can be used to replace the web service's native UI and take its role in terms of sending data to business layer and receive response from it.

2.2.3 Unit Testing

This approach uses a popular testing method called unit testing. "The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules."[3] Regarding the aspect of directing data, unit testing sends data to almost everywhere. A good analogy is that when launching a rocket. Every engineer is assigned to inspect the status of one tiny part of the rocket. Hundreds of them work together and simultaneously to finish inspecting the entire rocket. The following code is a JUnit test case:

```
// SimpleTest.java
import static org.junit.Assert.*;
import org.junit.Test;

public class SimpleTest {
	@Test
```

```
        public void test() {
                //Class Simple is being tested
                Simple tester = new Simple();
                //check if 1 + 2 + 3 returns 6
                assertEquals("1 + 2 + 3 must be 6", 6, tester.add(1,2,3));
        }
}
```

3 Comparison

It is not doubt that efficiency plays an important role when it comes to judging which approach is the best. Therefore all factors that influence the efficiency of the test should be analysed separately. The three approaches will be compared against each of these criteria. And in the end the criteria will be weighed based on their level of importance to produce one final sum for each approach.

3.1 Time

This subsection measures the average amount of time to run one single test. All three approaches will use the exact same inputs. The data used will be typical for VIP room clients. Due to confidentiality and copyright, the input data can only be briefly described as: It contains six mandatory fields and seven optional fields. Three of the mandatory fields are numerical fields and the rest are alphabetical. All seven optional fields are alphabetical fields.

Five sets of input was selected with each taking notably different amount of time when tested. Each set is tested three times on each approach to increase precision and the result is shown in Figure 5 (All values take the unit of a millisecond):

Figure 6 illustrates the average time taken every test case:

The graph shows that all three approaches finish all test cases in the same time frame with SOAP taking the longest and JUnit taking the least. This result is reasonable since the amount of time that runtime execution takes is dominant in the values displayed on the graph. At the mean time, the SOAP protocol wraps input data in an envelope when sending and checks the output with assertions and validations. This can explain the fact that it is taking longer than the

	UI	SOAP	JUnit
	642	764	655
	638	759	658
1	644	763	653
	850	954	841
	821	955	848
2	833	956	839
	1402	1587	1411
	1413	1574	1418
3	1399	1577	1417
	1322	1523	1355
	1342	1503	1342
4	1324	1508	1324
	4511	4622	4511
	4532	4636	4532
5	4568	4629	4568
Average	1749.4	1887.333	1758.133

Figure 5: Time taken by different approaches

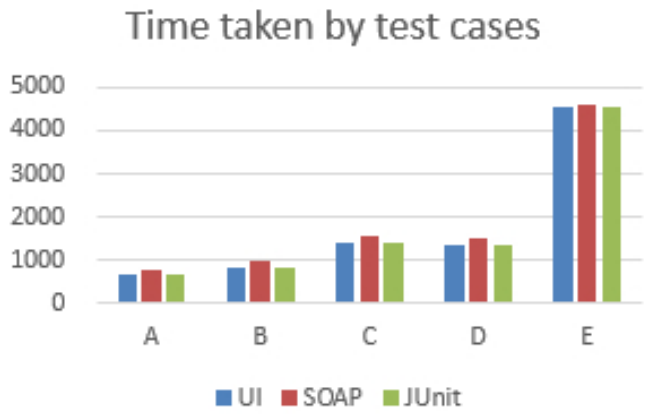

Figure 6: Time taken by test cases

others. Please note that this subsection does not measure the amount of time for quality analysts to enter the data when using the three approaches. This justified in the next two subsections.

3.2 Automation

Automation is the most popular topic among quality analysts who seek to improve their testing. Testing automation is a technique that boosts efficiency and reduces human labour. A large number of test cases are controlled by the system and run against the web application automat-

ically when a test command is generated by the tester or when the application is built.

The first approach is the least automation friendly approach, considering that data is entered from the UI. Although it is possible using a language such as Python, this is not encouraged or practically viable. The reason is that VIP room is an enterprise web application with thousands of customers. The security layer that wraps the application redundantly checks if the data is entered by some artificial intelligence to prevent brute force security penetration. Therefore automated testing is not feasible for this approach.

The possibility for automated SOAP testing solely depends on the software used. Some light weight SOAP software (or web client) does not have automation features. However in SoapUI, test cases are managed to be automation ready. It means that test cases are grouped and named so a particular group of cases can be extracted and run against the application instantly. The group and number of cases are controlled by the tester.

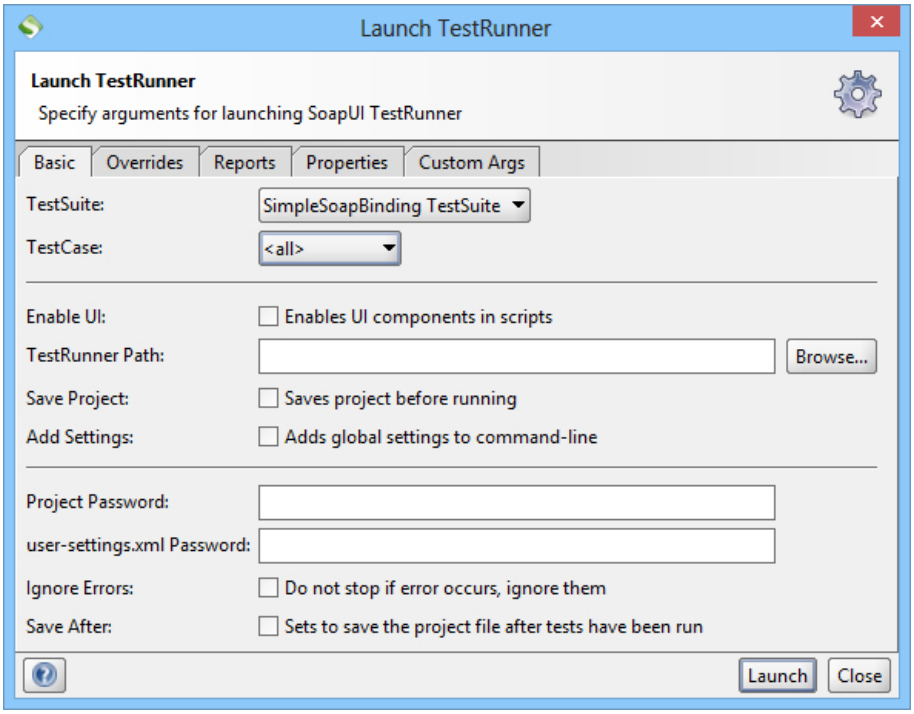

Figure 7: SoapUI automation feature

The window above is SoapUI's interface for automation testing. Although it does save the tester a lot of time and reduce the chance for mistakes, it is still not completely automated in the sense that it still needs to be configured and launched manually every time. True automation requires no human efforts after the initial test case configuration.

One of the advantages JUnit has is that it is integrated in the Maven build process. "Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information."[4] The VIP room web application uses Maven as its build manager. Every time a complete build (tests can be skipped to save build time) is done, the JUnit test cases are automatically run against the application. should any test fail, detail error message will be displayed in the console and the build will be marked as unsuccessful. As previously explained, JUnit "tests the smallest piece of testable software in the application."[3] This means that JUnit test cases are likely to be static as shown by the code block on page 6. This seems to defeat the definition of functional testing, which should cover all permutation of scenario. However in software engineering, there are techniques to prove that if all single blocks are performing correctly, the entire application will work in the right way. On the other hand, there are ways connect JUnit test cases to implement a functional test case. A test controller is needed to wrap the cases to form a dynamic test. With the modified version of Simple.java and SimpleTest.java listed in the appendix, functional testing can be achieved.

3.3 Reproduction

A good testing method should have easily maintained test cases as well as simple reproduction procedure. This subsection is going to assess the easiness for an approach to save and transfer test cases between software and computers.

This is a criterion that causes the first approach to disqualify as a viable solution for VIP room testing. Entering data from UI cannot be recorded or saved for easy reproduction. This is why many considers this approach as a casual and informal way to do functional testing. On the other hand, both SOAP and JUnit are quite competitive in this category. SoapUI exports its test projects, which contain test suites and cases, into XML files. These XML files can be read by many other SOAP clients such as SoapSonar to import the projects. Finally, JUnit test cases are merely java files which can be transferred between testers. JUnit does fall behind in terms of compatibility. Since the test cases are so closely written to the java files that build the web application, transferring test cases can cause version control conflict and dilemma.

The analysis from section 3.2 and 3.3 have justified that the average time for the tester to enter data cannot be taken into account directly. The lack of automation and reproduction for approach one makes the testers who use that method spending much more time entering data than those who choose the other two approaches.

3.4 Productivity

This last subsection analyses how much does each approach help the tester to find defects in the code and communicate the defect to developers. This is the part that JUnit takes the leading position. Due to its deeply integrated nature, JUnit is able to detect run-time errors and unexpected outputs instantly. Moreover, the built-in error messaging and logging feature displays and saves the error neatly for developers or future testers to refer to.
SOAP testing does not quite have the same power as JUnit in this category. Although it is able to detect and log unexpected output through assertions, all run-time error messages are completely out of its control. Instead they depend on how the web application itself handles these errors. Most of the time a registered error message will be displayed on the SOAP client and testers will have to refer to the application's design document to find more detail in order to pin point what went wrong. Finally, the first approach has the least amount of control over these errors since not even assertions can be implemented. The tester will have to have the expected output

on paper in order to determine if the output is legitimate.

4 Evaluation

A decision matrix will be used to determine which approach is the most suitable for VIP room. 100 points will be distributed to the four criteria analysed above and the points each criterion gets will be then distributed to the three testing methods. Among the four criteria, the order of impact on efficiency is Automation (40) > Reproduction (20) = Productivity (20) > Time (15). Based on the analysis above, the final distribution is:

Table 1: Decision Matrix

	Time	Automation	Reproduction	Productivity	Total
UI	5	0	0	2	7
SOAP	5	20	11	6	42
JUnit	5	20	9	12	46

5 Conclusions

The decision matrix indicates that the first approach trails in almost every category while the other two competes for only a couple of points to take the lead. All three methods perform relatively similar in Time since the runtime execution time takes most part of the time measured. SOAP and JUnit splits the forty points in Automation how powerful both methods are in this category. SOAP does have a slight edge over JUnit in maintaining and reproducing because the deeply integrated nature of JUnit is losing itself points in version controlling. However the latter was able to dominate in productivity with its great emphasis on small unit code quality which help it to secure the final lead against SOAP.

Although the decision is usually used to determine the best solution to a problem, it is important to notice that a quantitative value is produced as the final result. These values also shows how close each solution is with the others in terms of suitability. In this case, JUnit only beats SOAP by 4 points. This suggests that SOAP is not too bad of a solution for VIP room's testing. In fact, due to the substantially large difference in the two approaches' mechanics, it is most appropriate to use both methods at the same time. This thus can not only complement the disadvantage of both methods, but also creates redundancy to reduce the room for human error in testing.

6 Recommendations

Based on the analysis of the three different approaches to functional testing for VIP room, this report recommends both SOAP and JUnit testing to be used as a duo testing system. This compensates both the lack of control SOAP has over the individual functions and methods and the lack of thoroughness when using JUnit. Human errors can also be kept at minimum when this redundant testing technique is present.

JUnit test cases should be automatically run against the application on its build process, while the SOAP test cases shall be run when the application is deployed onto the server. SOAP test cases should be carefully labelled and grouped in order for quality analysts to share. JUnit test case modifications should be severely managed by version control software. This is because of the fact that a failed JUnit test fails the build process. Unwanted changes on tests can cause many co-workers to fail in building their application locally.

References

[1] Telus Health Inc., ”Telus Health: Home,” Telus Health Inc., *http://www.telushealth.com/* (current April 11 2014).

[2] W3C Note, ”Simple Object Access Protocol (SOAP) 1.1,” W3C Note, *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/* (current April 11 2014).

[3] Microsoft Corp., ”Unit Testing,” MSDN - Microsoft, *http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx* (current April 11 2014).

[4] Apache Foundation., ”Maven,” Apache Foundation, *http://maven.apache.org/* (current April 11 2014).

Acknowledgements

I would like to thank my manager Kevin Ho and Quality Analyst Anshuman Ghandi, both of whom gave inputs on how functionality testing is done in other projects produced by Telus Health. Especially Anshuman who gave me a thorough introduction in SOAP testing and provide me with testing tools such as SoapUI and smaple test cases.

I would also like to thank Anna Chen, a friend of mine who inspired me to write on this topic and encouraged me to finish this report on time.

Appendix A Sample WSDL file

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wsdl:definitions xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://service" xmlns:intf="http://service"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://service">
 ▼<!--
    WSDL created by Apache Axis version: 1.4
    Built on Apr 22, 2006 (06:55:48 PDT)
  -->
 ▼<wsdl:types>
  ▼<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://service">
   ▼<element name="add">
    ▼<complexType>
     ▼<sequence>
        <element name="a" type="xsd:int"/>
        <element name="b" type="xsd:int"/>
        <element name="c" type="xsd:int"/>
       </sequence>
      </complexType>
     </element>
   ▼<element name="addResponse">
    ▼<complexType>
     ▼<sequence>
        <element name="addReturn" type="xsd:int"/>
       </sequence>
      </complexType>
     </element>
   </schema>
  </wsdl:types>
▼<wsdl:message name="addResponse">
   <wsdl:part element="impl:addResponse" name="parameters"></wsdl:part>
  </wsdl:message>
▼<wsdl:message name="addRequest">
   <wsdl:part element="impl:add" name="parameters"></wsdl:part>
  </wsdl:message>
▼<wsdl:portType name="Simple">
  ▼<wsdl:operation name="add">
     <wsdl:input message="impl:addRequest" name="addRequest"></wsdl:input>
     <wsdl:output message="impl:addResponse" name="addResponse"></wsdl:output>
   </wsdl:operation>
  </wsdl:portType>
▼<wsdl:binding name="SimpleSoapBinding" type="impl:Simple">
   <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  ▼<wsdl:operation name="add">
     <wsdlsoap:operation soapAction=""/>
    ▼<wsdl:input name="addRequest">
       <wsdlsoap:body use="literal"/>
     </wsdl:input>
    ▼<wsdl:output name="addResponse">
       <wsdlsoap:body use="literal"/>
     </wsdl:output>
   </wsdl:operation>
  </wsdl:binding>
▼<wsdl:service name="SimpleService">
  ▼<wsdl:port binding="impl:SimpleSoapBinding" name="Simple">
     <wsdlsoap:address location="http://localhost:8080/test/services/Simple"/>
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 8: Simple.wsdl

Appendix B Simple.java

```
// Simple.java
public class Simple {
	public int add(int a, int b){
		return a + b;
	}
	public int subtract(int a, int b){
		return a - b;
	}
	public int multiply(int a, int b){
		return a * b;
	}
}
```

Appendix C SimpleTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;
import service.Simple;

public class SimpleTest {
	@Test
	public void testAdd() {
		//Class Simple is being tested
		Simple tester = new Simple();
		//check if 1 3 returns 4
		assertEquals("1 + 3 must be 4", 4, tester.add(1,3));
	}
	public void testSub() {
		//Class Simple is being tested
		Simple tester = new Simple();
		//check if 8 4 returns 4
		assertEquals("8 - 4 must be 4", 4, tester.subtract(8,4));
	}
	public void testMul() {
		//Class Simple is being tested
		Simple tester = new Simple();
		//check if 2 5 returns 10
		assertEquals("2 * 5 must be 10", 10, tester.multiply(2,5));
	}
}
```