Reducing cost in regression testing of web service

Conference Paper · September 2012

DOI: 10.1109/CONSEG.2012.6349498

CITATIONS
10

READS
1,747

1 author:

Animesh Chaturvedi
Indian Institute of Information Technology, Design and Manufacturing Jabalpur
17 PUBLICATIONS **112** CITATIONS

Reducing Cost in Regression Testing of Web Service

Animesh Chaturvedi
Indian Institute of Information Technology, Design and Manufacturing Jabalpur
Email: animesh@iiitdmj.ac.in

***Abstract*- Regression testing and retesting of a modified Web Service can be very costly as it tends to generates large number of test cases. Regression testing cost can be reduced significantly by identifying and testing the modified portion of the Web Service only. This avoids the costly construction of new test cases and the unproductive rerunning of existing test cases when it can be guaranteed that the unmodified code of web service will produce the same results as it produced previously.**

In this paper, by designing Web service graph we propose an effective method of identifying the modified areas (in this paper area means code of web service to be tested) and use this information to reduce regression testing efforts. In this process, we define and use a set of metrics to categorize the testing of Web service. We demonstrate the applicability of the proposed approach using a case study that consisted of two steps. In the first step, we compared two versions of WSDL's to identify the inserted, modified and deleted areas of the Web services. Next, we use this information to generate the reduced test cases.

Keywords: Regression testing, web service, XML Diff, WSDL.

I. INTRODUCTION

The W3C defines a "Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network" [3]. It has an interface described in a machine processable format (specifically, Web Services Description Language, known by the acronym WSDL). WSDL is a XML-based language that is used for describing the functionality offered by a web service [4]. A WSDL description of a web service (also referred to as a WSDL file) provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. It thus serves a roughly similar purpose as a method signature in a programming language.

Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web related standards.

With the success of service oriented architecture, developers are taking Web service in their important concern. Lots of other work is going on Web service domain like Grid Computing, Cloud Computing. For example a technical report for Grid services deployment [24] on Grid environment as described in [23]. Another example scalable cloud can be used for efficient Web Service Testing process [7]. Moreover, Cloud systems offer services at two different layers: Infrastructure Layer, Virtual Machine Layer. Usually the Infrastructure Layer is designed according to the Service Oriented Architecture (SOA) model, exploiting web services technology [8]. In order to manage their platform with a service interface they adopted WSAG, a framework that implements the WSAI (Web Services Agent Integration) project. It supports the communication between Web Services and agents [8]. An excellent example of WSDL usage is eucalyptus famous deployed cloud computing software private platform [21]. Eucalyptus uses WSDL as given in the source code of eucalyptus on Github [22].

For above reasons we require good testing techniques of Web service. A web service can be a standalone (or a part of) web application that can be independently developed and deployed. Like any application, when modified, the web service needs to go through the testing changes made to it – i.e. regression testing. Regression testing involves selective retesting of an application or component to verify that changes have not caused any unintended affects and that the application or component still complies with its specified requirements. Provide sufficient confidence that the changes were correct [2].

Regression testing of a web service can be performed with the help of WSDL [1]. Similarly one can also use XSD for the regression testing of web services. Problem with WSDL usage is that humans are very uncomfortable to manipulate WSDL. So WSDL usage is typically done in automated form i.e. with the help of SOAP posting which is cumbersome and error-prone. There are some tools available to send SOAP request and to get SOAP response in automated way like WebInject [16] and soap-UI [17]. But their support to regression testing of web service is not good. Therefore there is a requirement of automate the regression testing of web service.

In this paper, we present a technique of regression testing that can be adopted as a tool to perform automated regression testing of web services. Now, describing a method of identifying modified area of Web Service. Because of discrimination among modified and unmodified areas of Web Service can be retested with lesser number of test cases. Use NetBeans IDE as a tool to manipulate WSDL and XSD for the manual regression testing of Web service which can be converted to an automated tool. NetBeans Diff [5] for identifying the difference i.e. changes in between WSDL's for two versions Web service. Its output format is same content as returned by other Diff tools and Diff XML [6]. Result of the difference is shown by the color i.e. light blue color for modified code, light red color for deleted code, light green for inserted code.

The rest of the paper is as follows. Section II describes about related work of the presented approach i.e. about previous works in Regression Testing of Web services. In Section III we are going to define five testing categories for

Web Service testing. Section IV contains our approach of reduction in the cost of Regression testing of Web Service. In Section V algorithm on reduction in the cost of Web Service regression testing is written. To prove this work we performed a case study on two different versions of our Web service in Section VI. In the end, Section VII contents conclusion i.e. reduction in the cost of regression testing of Web service.

II. RELATED WORK

Till now work in regression testing of Web service is less because it requires good knowledge of both Regression testing and Web services. But we can see many good contributions in the domain of Web service testing and regression testing individually. Regression Testing of Web Service is yet to be explored further. In 1995 David Binkley [9] describe about reducing the test cost of regression testing by an algorithm that uses semantic differences and similarities between old and new programs, but this paper is only for cost reduction of regression testing not for web service testing. In IEEE transaction [19] author presented the controlled experiments to examining test suite granularity and test input grouping on the several methodologies: retest-all, regression test selection, test suite reduction, and test case prioritization.

Four metric for Web service testing with the help of WSDL [1] and a modified form of metric in [10] used for automated test case generation but we extend these metric and defining in better way in Section III. In 2009 author Tamim and Reiko proposed a model-based approach to solve problem Regression Testing of Web Services [20]. This models describes interfaces of services before and after the change are compared in order to analyze the evolution of a system, identifying which tests need to be rerun and where new ones are required.

Automated test cases of Web Service can be generated as described in [10] with the help of WSDL, first it parse WSDL and transformed it into the structured DOM tree then generate test cases as report. In paper [18] authors presented an approach and a tool to allow users to run a test suite against a service to discover if functional and non-functional expectations are maintained over time.

In [11] a safe regression testing algorithm that selects an adequate number of non-redundant test sequences aiming to find modifications related errors is mentioned with an algorithm. The algorithm selects every test sequence that corresponds to a different behavior in the modified system. Another approach to apply a safe regression test selection technique to Web services is given by Rothermel and Harrold [12] with the goal to develop a regression test selection and regression testing framework that supports integrated software change management system. But their case study is based on very small and incomplete Web service, even more they are not taking help of WSDL for identifying the testing areas as mentioned in [1] and used in [11], [10].

Analysis of Web service testing tool (SoapUI outperforms PushToTest, and WebInject) [13] with several selected web services is done but in those tool there is no efficient approach of regression testing till now.

III. METRICS OF WEB SERVICE TESTING

Performing regression testing of Web service, it is requiring manipulating information provided by WSDL. We are modifying previous metric [1] in more precise, formal and well-defined way for Web service testing. Following are the Web service testing metric based on the WSDL information.

T1 *Web Service Policy Testing (WSPT):* It is a non functional testing in which one can test behavior of a policy assertion which represents a requirement, a capability, or other property like binding security policy assertion. MIME media type name of application, MIME subtype name, Required parameters, Optional parameters, Encoding considerations, Security considerations, Interoperability considerations, Published specifications, Applications which use this media type, Additional information (File extension, Fragment identifiers, Base URI, Macintosh File Type code, Intended usage, Author/Change controller) [14].

T2 *Message Part Testing (MPT):* User of WSDL must be able to post Soap message correctly and get the Soap response correctly for that we required testing message part communication.

T3 *Operation Code Testing (OCT):* It is a functional testing based on the operation logic we want to provide to the user of WSDL. Logic of operation must be tested thus proper output will be generated by Web service.

T4 *XSD Input - Output Testing (XSDIOT):* It is important functional testing and performed to provide correct input and get output correctly from the service. It can be done by manipulating XSD of web service. It is of three type Input dependency type, Output Dependency type and Input-Output Dependency type.

T5 *Web Service Calling Testing (WSCT):* It might be possible that Web service is combined to another component of web world i.e. one Web service is calling other Web service, Website etc. Therefore testing of component calling by web service is also required.

Type of testing's mentioned above are used in paper by using abbreviation **T1 WSPT**, **T2 MPT**, **T3 OCT**, **T4 XSDIOT**, and **T5 WSCT** respectively further in the paper.

IV. REGRESSION TESTING OF WEB SERVICE

Any XML tools contains feature of XML diff through which one can get the differences in between two XML file (old and new). We used NetBeans IDE for Diff to show the differences in between old and new WSDL and used those differences in WSDL to perform regression testing on our web service manually; this technique can be converted to automated regression testing tool. Inside NetBeans Tools→Diff is the feature in which we can compare two files.

Find reusable test case areas to reduce the cost of retesting. Perform following step for the comparison of two versions of WSDL of a web service (first version SAAS_1 and second version SAAS_2).

1) Generated WSDL's of old and new versions of web service.
2) Then apply XML Diff to get the difference between two WSDL in Fig. 1.
3) Finally we get the result in the differences we made in first version of Web service to get second version of Web service.

Anyone can notice the following differences in the two versions of Web Services with the help of difference in colors.

1) In Fig. 1 web service policy modified and deleted is noticed i.e. it requires testing of type T1 (WSPT).
2) In Fig. 1 two messages posting part is inserted i.e. it requires testing of type T2 (MPT) because of two new operations ("Searching" and "readingFile") in Fig. 1 is shown i.e. it requires testing of type T3 (OCT) .
3) In Fig. 1 XSD is also modified we further applied XML Diff to those XSD's. Fig. 2 i.e. it require testing of type (XSDIOT).
4) In Fig. 2 new elements are added "Searching", "SearchingResponse", "readingFile" and "readingFileResponse" i.e. it requires testing of type T4 (XSDIOT).
5) In Fig. 2 T4 (XSDIOT) name of element "Index" is modified to "Indexing" in operation "Index".
6) In Fig. 2 T4 (XSDIOT) input and output dependency of operation "Index" is also changed. String input type is changed to integer input type. And string output type is also modified to integer output type. With the help of these input and output dependencies we can make modifications in soap posting program in which we can put localName as the name of element.
7) With the help of Fig. 2 T4 (XSDIOT) we can also get the input and output dependencies of "Searching" and "readingFile" operations each of those were string type.

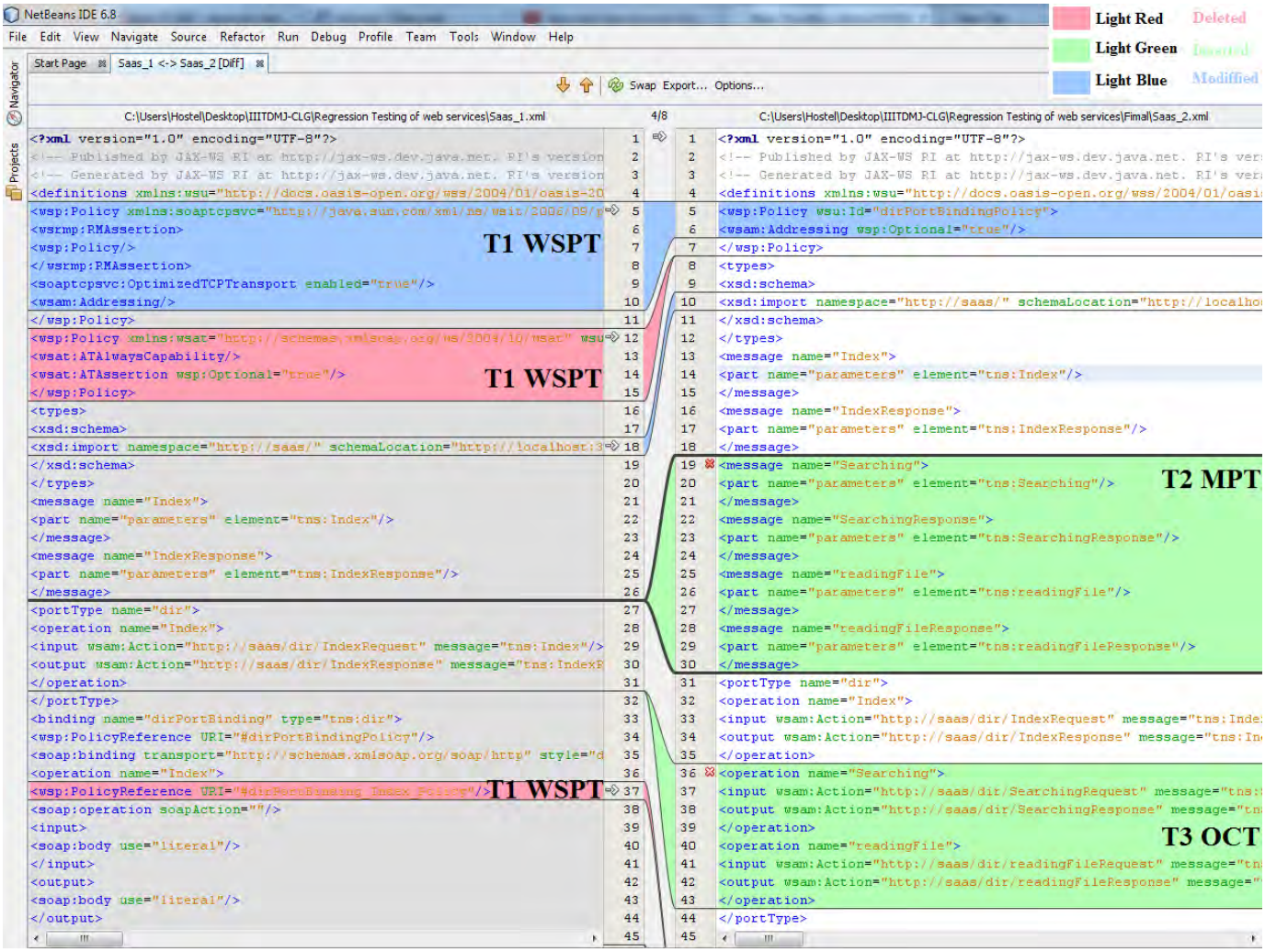

Fig. 1. XML diff of old WSDL (left) with new WSDL (right). Difference between 2 WSDL can be seen in added line of code highlighted by light green color; deleted lines of code highlighted by light red color and modified lines by light blue color.

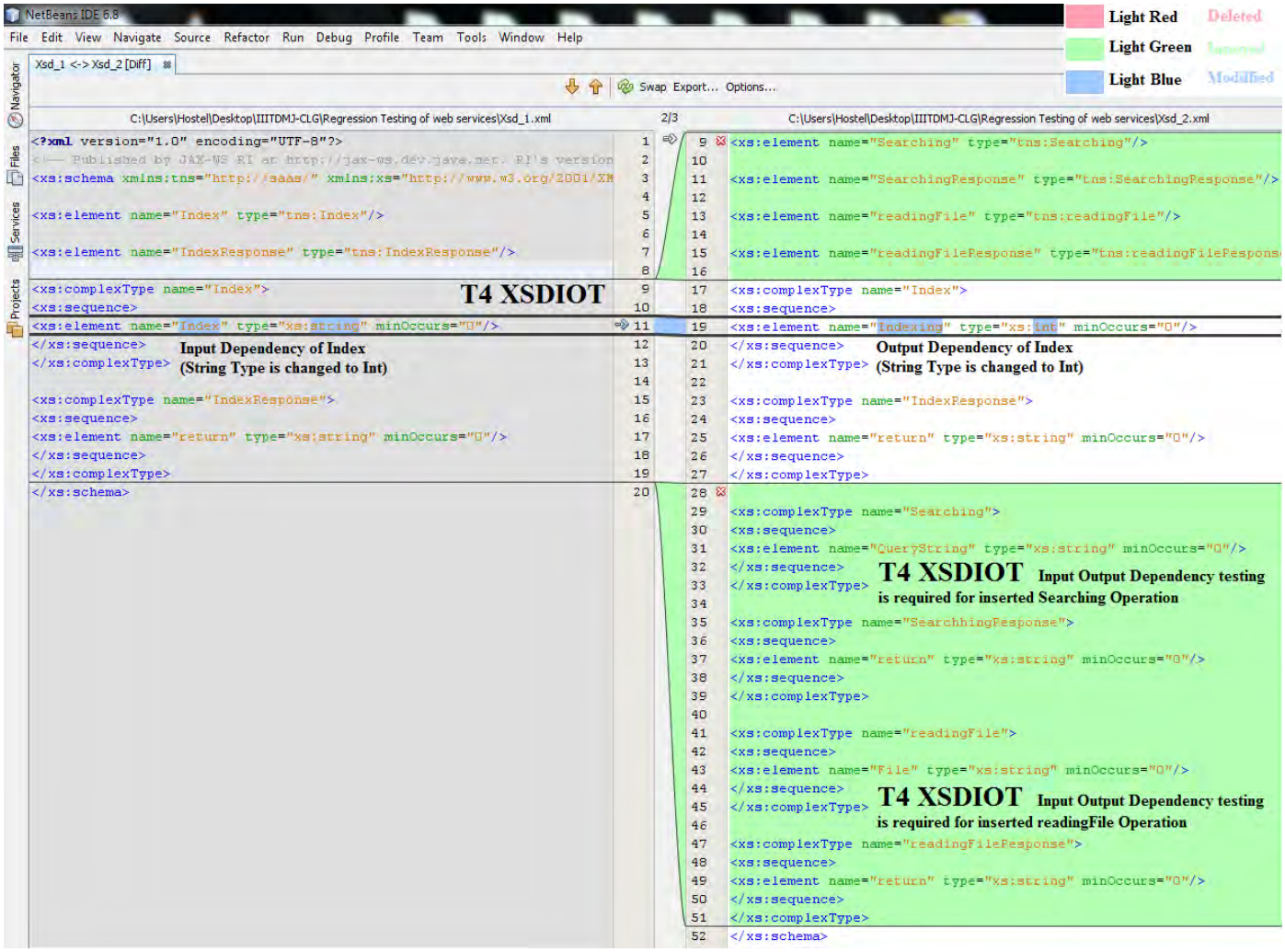

Fig. 2. XSD Diff between Saas_1 and Saas_2 i.e.T4 (XSDIOT) testing type.

We can make Web service graph (WSG) of a Web service by using WSDL and XSD as shown below in Fig. 3. We can construct WSG by describing policy of Web service as root, operations as child of root, and input/output mentioned in XSD as leaf. With above interpretation of differences in XML Diff we construct Web service Graph (WSG) diff in SaaS_1 and SaaS_2 given below in Fig. 3

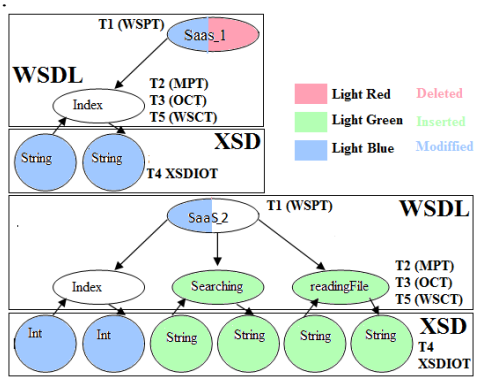

Fig. 3. Web Service graph of Saas_1 and Saas _2

Deletion by light red color, insertion by light green color modifications by light blue color is highlighted in (WSG).

V. REDUCTION IN THE COST OF TESTING

Apply reduced regression test cases algorithm on taken two versions of web services i.e. Saas_1 and Saas_2 to get reduced test cases as the output. Then we will validate the algorithm and the approach of our Reduction in the cost of Regression testing technique by a case study. Operation is an important part of web service because business logic is coded in it. Hence it is in special consideration of algorithm. Let us consider

WS – Web service version 1 Saas_1
*WS** – Modified Web service version 2 Saas_2
T-old= Test Cases for the code of WS
T-new = Test Cases for the code of WS*
*T** = Reduced test Cases
Saas_1WSDL = WSDL of Saas_1 in XML
Saas_2WSDL = WSDL of Saas_2 in XML
PartofWSDL = Output format is the **insert, modified, delete** define above in Section II**.**

Functions Apply *NetBeans Diff* (Saas_1WSDL, Saas_2WSDL)
{//Algorithm of Diff Xml
Return *PartofWSDL*
}

Function ReducedRegressionTestCase (*PartofWSDL*, *T-old, T-new*)
{

Assignment

1. *T* = T-old* //Reduced test case i.e. T* must have T-old initially, then we can reduce test cases from T-old.

2. *Del*=word, group of line of **deleted operation/policy** of WS in *Saas_1WSDL* (example in Saas_1 "policy delete ")
3. *Ins*=words, group of line of a **inserted operation/policy** of WS* in *Saas_2WSDL* (example in Saas_2 "Searching")
4. *Mod*= words, group of line of modified part of **operation/policy** of WS in *Saas_2WSDL* (example "Index")

5. *td:* {*td* ∈ *T-old* | *td* = Test cases of a **deleted** operation or policy of WS}
6. *ti:* {*ti* ∈ *T-new* | *ti* = Test cases of a **inserted** operation or policy of WS*}
7. *tm:* {*tm* ∈ *T-new* U *T-old*| *tm* = Selective test cases of a **modified** part of operation or policy of WS to make WS*}
8. *tu:* {*tu* ∈ *T-old* | *tu* = Test cases of a **unmodified** operation or policy of WS}

9. **For (**run till the end of WSDL of Saas_1**)** **//** scan the WSDL of Saas_1 from starting till its end
   {
   if(*PartofWSDL==Del***)** //search for the **deleted** operations or policies of WS
   T = T* - td* //reducing test cases of **deleted** operations or policy
   }

10. **For (**run till the end of WSDL of Saas_2**)** **//** scan the WSDL of Saas_2 from starting till its end
    {
    if(*PartofWSDL==Ins***)** //search for the **inserted** operations or policies of WS*
    T = T* + ti* //inserting test cases of **inserted** operations or policy

    if(*PartofWSDL==Mod***)** //search for the **modified** operations or policies of WS in WS*
    T = T* + tm* //inserting test cases of **modified** operations or policy
    }

11. *T* = T* - tu* //reducing test cases of **unmodified** operations or policy
12. **Return** *T** //Return reduced test cases i.e. *T**

}

First of all store the output (*PartofWSDL)* of Netbeans diff applied to our Saas_1 WSDL and Saas_2 WSDL. Secondly, pass the *PartofWSDL to the ReducedRegressionTestCase* function. Description of above algorithm is as follows

1) Line number **1** *T* = T-old:* Reduced test case initially has old test cases.
2) Line number **2-8** was declarations of inserted, deleted modified, unmodified areas and their test cases for Web Service.
3) Line number **9 For (**run till the end of WSDL of Saas_1**)** scanning of the WSDL of Saas_1 from starting till its end, to identify the number of lines, words that are deleted. Then test cases corresponding to those deleted lines/words for Web service code should be deleted.
4) Line number **10 For (**run till the end of WSDL of Saas_2**)** scans the WSDL of Saas_2 from starting till its end, to identify the area i.e. line, words that are inserted or modified. Then add test cases for the new areas and add selective test cases for the modified areas.
5) Line number **11** ***T* = T* - tu*** will delete all the remaining unused test cases which are already executed in the testing of previous version of web service. These test cases are not required because component of those test cases in web service are already tested with these test cases.
6) Line no. **12 Return** ***T****(reduced test case) value is calculated by applying algorithm is evaluated to be *T* = ti + tm -td – tu*.

VI. CASE STUDY

For our case study we again go to modify Saas_2 to Saas_3 by doing following activities.

1) Changing Web service policy by deleting addressing in Saas_3.
2) Inserting new operation "editFile" to Saas_3.
3) Changing both input and output type of "Index" operation from String to Integer.

To show the implications of our approach we take WSDL's of two versions of our Web service i.e. Saas_2 and Saas_3. Again, perform steps given in section IV for diff in between Saas_2 and Saas_3. Generate differences between two WSDL's is shown in Fig. 4 and in between XSD's of Saas_2; Saas_3 is shown in Fig. 5. Anyone can notice following differences in Saas_2 and Saas_3.

1) In Fig. 4 policy is deleted from Saas_2 to Saas_3 highlighted in light red color i.e. T1 (WSPT).
2) In Fig. 4 inserted message posting part of operation "editFile" i.e. T2 (MPT) testing type.
3) In Fig. 4 insertion of operation in "editFile" i.e. T3 (OCT) testing type.
4) In Fig. 4 modification in XSD of Saas_2 and Saas_3 i.e. T4 (XSDIOT) testing type.
5) In Fig. 5 modification in input/output type of operation "Index" i.e. T4(XSDIOT) testing type
6) In Fig. 5 inserted operations "editFile" must be tested for input output dependency i.e. T4 (XSDIOT) testing type.

In Fig. 4 and Fig. 5 identified testing areas of Web service Saas_3 highlighted by colors. After identifying the reduced testing areas in Web service Saas_3 from above differences of WSDL's and XSD's. Now, we are going to describe those testing areas in the code of Web service Saas_3. This is done by giving example for operational code testing (OCT), XSD input-output testing (XSDIOT), and message part testing (MPT) in this case study of above given approach of regression testing.

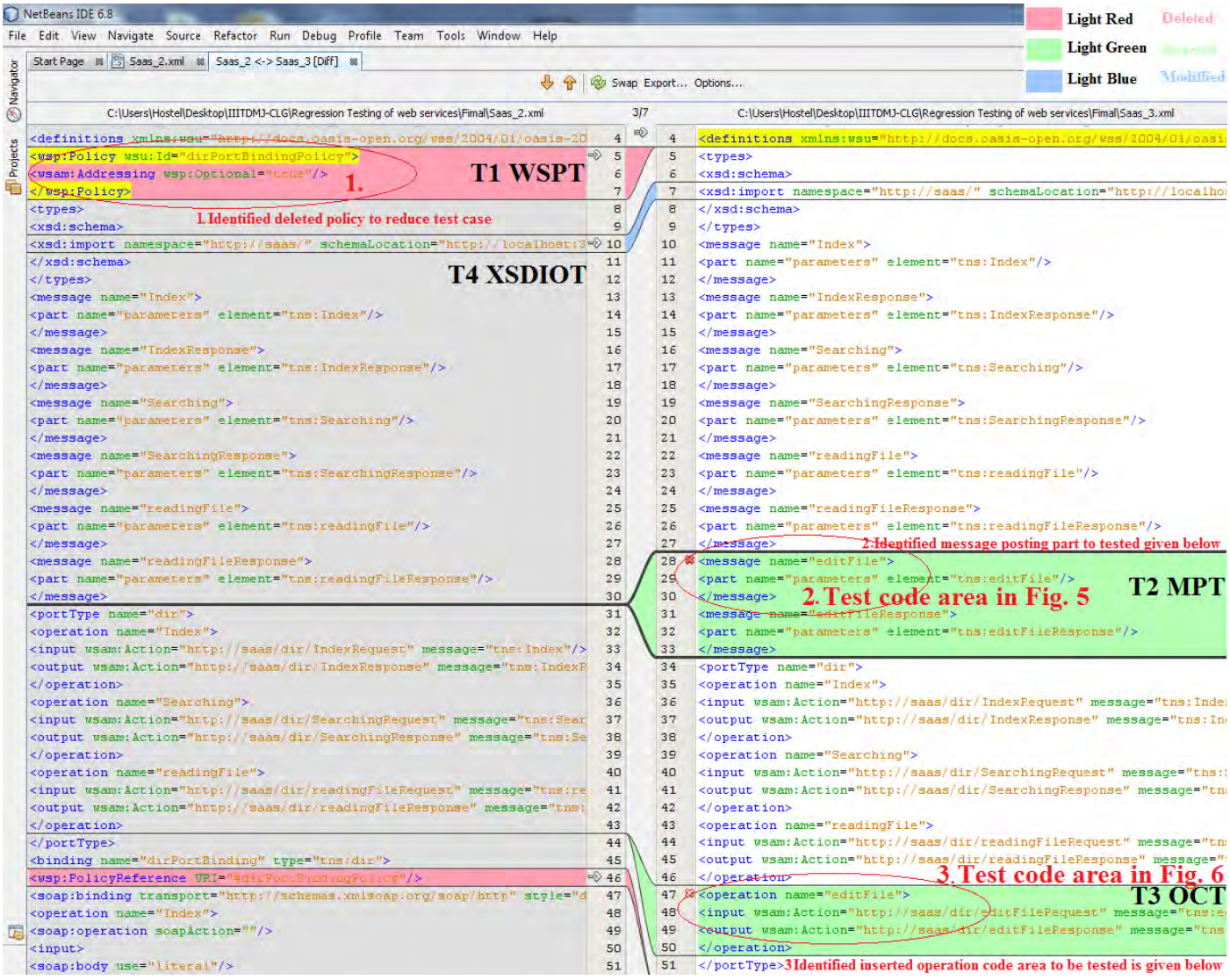

Fig. 4. Deletion of policy is highlighted in light red color, insertion of operation and message part of "editFile" is highlighted in light green color and modification of XSD is highlighted in light blue color.

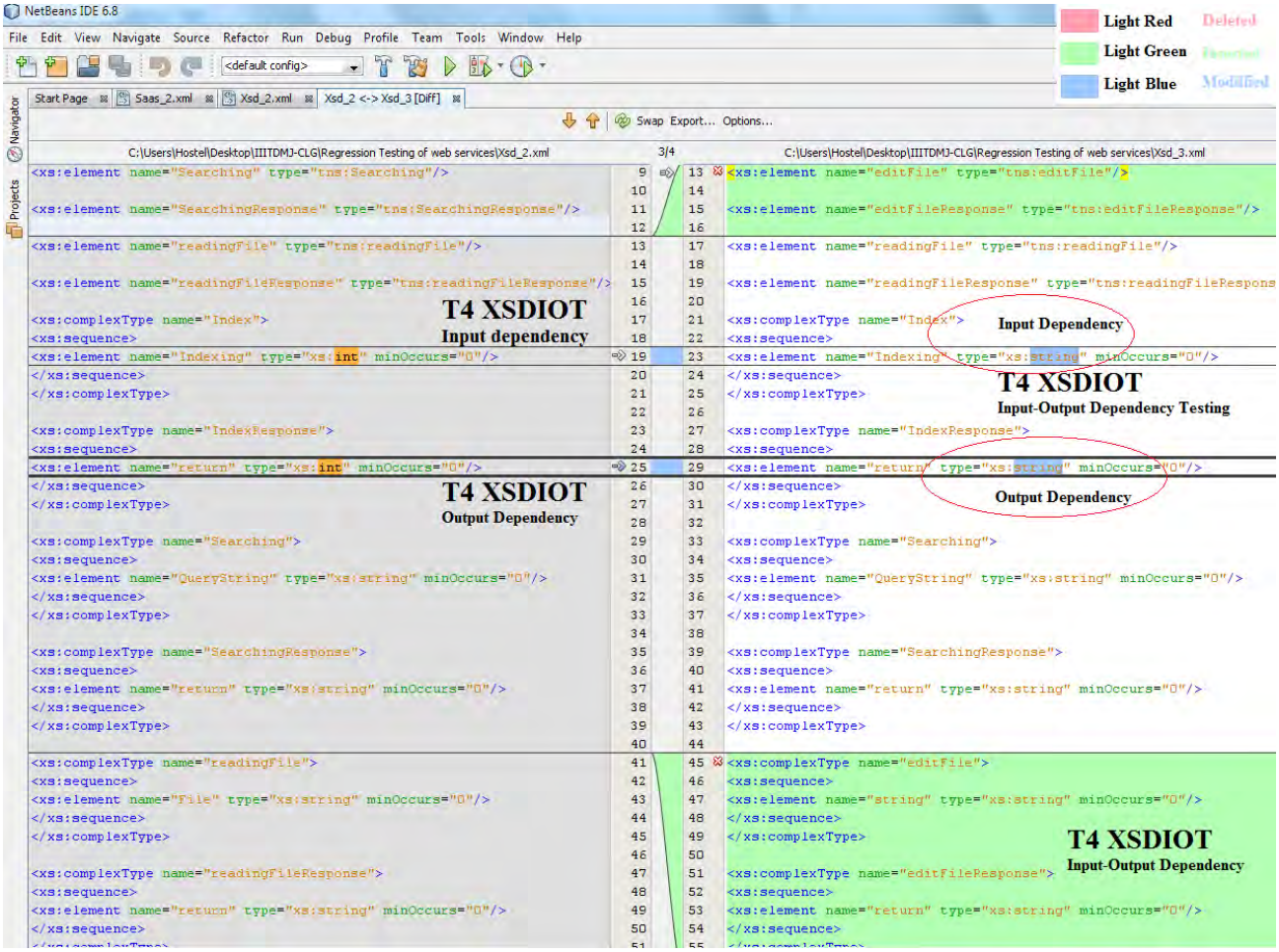

Fig. 5. XSD Diff in between Saas_2 and Saas_3, input type and output type is modified in "Index" and newly inserted "editFile".

T2. Message part testing: Code area to be tested for the message posting part identified in Fig. 4 i.e. T2 (MPT) is shown in Fig. 8 (in next page). By testing Soap Posting one can cover testing of message posting area of Web service.

T3. Operation code testing: Operation "editFile" is inserted operation so it must be tested. In Fig. 6 operation code for "editFile" identified in Fig. 4 i.e. T3 (OCT) is described. By testing this newly inserted code area in Web Service Saas_3 we can check the functionality of operation "editFile".

```
/**
 * Web service operation author Animesh Chaturvedi
 */
@WebMethod(operationName = "editFile")
public String editFile(@WebParam(name = "inputPosting")
String string) throws IOException {
    //TODO write your implementation code here:
    FileOutputStream fos = null;
try {
    String fileName = "C:\\Users\\ani\\Desktop\\Anand Pateriya.txt";
    fos = new FileOutputStream(fileName);
    fos.write(string.getBytes("UTF-8"));
} catch (IOException e) {
    throw e;
}
    return null;
}
```

Fig. 6. Testing area identified in Fig. 4 as T3 (OCT)

T4. XSD input-output testing: Two testing areas are identified in Fig. 5. First, operation "Index" is modified, so it must be tested in Web Service Saas_3. Testing area identified in the Fig. 5 is shown in Fig. 7 below. We can reduce testing in Web Service Saas_3 by only testing the input and output related code of "Index" operation.

Secondly identified area in Fig. 5 is newly inserted operation "editFile". For this code area is given in Fig. 6, for which complete editFile code must be tested because completely new operation editFile is inserted as identified in Fig. 5.

```
 * @author ani
 */
@WebService()
public class dir {
    @WebMethod(operationName = "Index")
    public String Index(@WebParam(name = "Indexing")
    String Indexing){
      File X      = new File(Indexing);
    new visitAllDirsAndFiles(X);
return "end";
}
```

Web service operation required to be tested because both input & output type is change

Output Dependency is changed to String Type

Input Dependency is changed to String type

Input-Output Dependency Testing Area

Fig. 7. Testing area identified in Fig. 5 as T4 (XSDIOT) i.e. modified operation "Index.

```
475         String s = jTextField2.getText().toString();
476     try {
477         SOAPConnectionFactory soapConnFactory = SOAPConnectionFactory.newInstance();
478         SOAPConnection connection = soapConnFactory.createConnection();
479         MessageFactory messageFactory = MessageFactory.newInstance();
480         SOAPMessage message = messageFactory.createMessage();
481         SOAPFactory soapFactory =  SOAPFactory.newInstance();
482         SOAPPart soapPart =  message.getSOAPPart();
483         SOAPEnvelope envelope = soapPart.getEnvelope();
484         SOAPBody body =  envelope.getBody();
1.       Name bodyName = soapFactory.createName("editFile", "ns1", "http://saas");
486      SOAPBodyElement bodyElement =  body.addBodyElement(bodyName);
2.       bodyElement.addChildElement("inputPosting").addTextNode(s);
488      message.saveChanges();
489      System.out.println("\nREQUEST:\n");
490      PrintStream tw = new PrintStream(new TextAreaOutputStream(jTextArea1));
491      System.out.println();
492      String destination = "http://172.27.217.7:38462/SAAS_initial/dirService";
493      SOAPMessage reply = connection.call(message, destination);
494      System.out.println("\nRESPONSE:\n");
495      TransformerFactory transformerFactory = TransformerFactory.newInstance();
496      Transformer transformer = transformerFactory.newTransformer();
497      Source sourceContent = reply.getSOAPPart().getContent();
498      tw.flush();
499      jTextArea1.setText("");
500      StreamResult result = new StreamResult(tw);
501      transformer.transform(sourceContent, result);
502      System.out.println();
503      connection.close();
504  } catch(Exception e) {
505      System.out.println(e.getMessage());
506  }
```

1. Message part name in Fig. 4 needed to be posted in Soap message for operation "editFile" using soapFactory class in local name method createName

2. For child element node name "inputPosting" of "editFile" in Fig. 5 we are post bodyName created by soapFactory for child element "inputPosting" above.

Fig. 8. Test area of message posting code of the web service identified in Fig. 4 as T2 (MPT).

Similarly also perform Web Service calling Testing and policy testing in web services.

Generation of WSG for Saas_2 and Saas_3 can be done based on the information discovered above. WSG of Saas_2 and Saas_3 is given in Fig. 9.

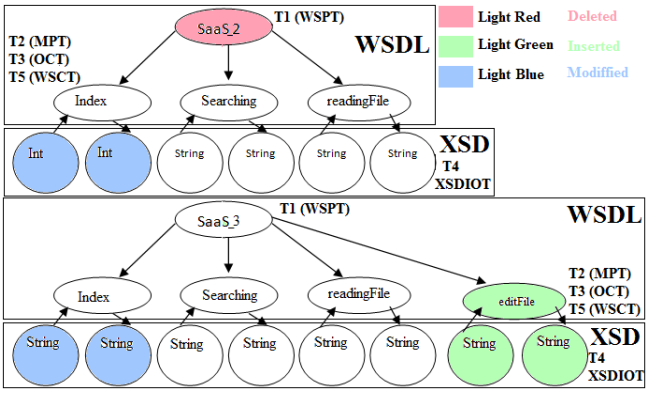

Fig. 9. WSG describing deletion of policy in light red color, insertion of "editFile" in light green color and modification of input-output type of "Index" in light blue color.

We verifying the efficiency of algorithm by doing cost reduction evaluation based on the WSG's of our case study. It can be logically inferred that "Number of test cases will get reduce as the testing area is reduced" because number of test case is directly proportional to testing area. So use testing area for ease of understanding.

Calculation for reducing cost in regression testing of Web service:

Total number of testing area to test Saas_3 without using above mentioned approach is **13**. Whereas with using reduced test case algorithm total number of testing area required is **5** in Fig. 9 colored areas in Saas_3. Explanation about number of testing areas i.e. relating to the above mentioned algorithm Reduced Test Cases

Initially, $T^* = T\text{-}old$

1) Policy assertion is deleted so test cases of this area should be removed it is *td* mentioned in algorithm is subtracted i.e. $T^* = T^* - td$.
2) "editFile" (input string, output string) is inserted so new test cases should be inserted it is *ti* mentioned in algorithm is added i.e. $T^* = T^* + ti$.
3) "Index" (input string, output string) add modify test cases accordingly it is *tm* mentioned in algorithm is added i.e. $T^* = T^* + tm$.
4) "Index", "Searching", "readFile", Saas_3 policy and I/O of Searching and readingFile is unmodified so test cases for these testing areas must be subtracted it is *tu* mentioned in algorithm is subtracted i.e. $T^* = T^* - tu$.

As a result percentage reduction in testing with respect of area is **5/13*100= 46 %**.

VII. CONCLUSION

Using diff gives us the differences in our web service versions, with which one can easily see that deletion, insertion

and modification in operations/policies to our Web service. Further build appropriate test suite for regression testing of Web services using reduced test case algorithm. Hence this reduced our testing effort. Summary of paper is given in Fig. 10.

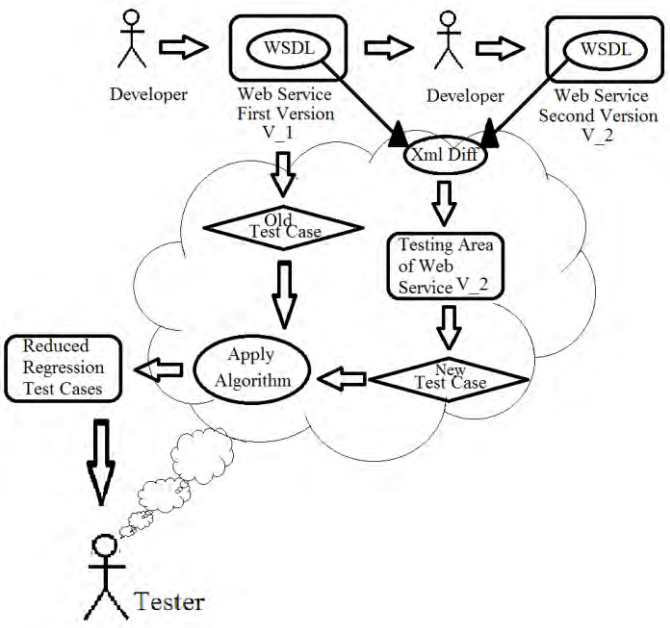

Fig. 10. Tester can identify testing areas in Web service and generate reduced regression Test cases using algorithm.

Percentage reduction in test area for our case study is 46%. As the size of Web service increases, percentage of test area reduces significantly because number of unmodified areas increases significantly.

NeatBeans diff works well with our Web service case study and we had not checked every aspects of Diff tool so it might be possible that it may create some problem. But it can definitely work fine with XML Diff. And source code of XML Diff can be easily manipulated and small modification in code can make it according to our requirement for regression testing of Web service thus this can solve our problem.

With above testing technique a new tool can be created for regression testing of web service or this new feature can be added to the existing tools of Web service testing. XML diff code is required for this method which is already available as open source therefore anybody can create an automated tool from our technique of regression testing of Web Service. We will try to develop this type of automated tool as our future work.

ACKNOWLEDGMENT

The author expresses a deep sense of gratitude to Late Dr. Manohar Chandwani who had motivated him to take challenges in higher computation.

REFERENCES

[1] W. T. Tsai, Ray Paul, Yamin Wang, Chun Fan, and Dong Wang, "Extending WSDL to Facilitate Web Services Testing", in Proc Seventh IEEE International Symposium on High Assurance Systems Engineering (HASE'02), pp. 1-2, 2002.

[2] Anjaneyulu Pasala and Ravi Gorthi "Emerging Trends in Software Test Plan*",* *http://www.csi-india.org/c/document_library/get_file?uuid=f7a765ae-45f4-4fc3-bbff-e1d11a720a38&groupId=10616*, August 8, 2012.

[3] Hugo Haas, Allen Brown, "Web Services Glossary," W3C *http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211*, April 30, 2012.

[4] "Web Services Description Language," *http://en.wikipedia.org/wiki/Web_Services_Description_Language*, July 13, 2012.

[5] Josef Pavlicek, "DIFF Specification," *http://ui.netbeans.org/docs/ui/DiffSpecification/DIFFSpecification.html*, April 30, 2012.

[6] Luuk Peters, "Change Detection in XML Trees: a Survey" in 3rd Twente Student Conference on IT, University of Twente, June 2005.

[7] Tharam Dillon and Chen Wu and Elizabeth Chang "Cloud Computing: Issues and Challenges," in Proc. 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 27-33, 2010.

[8] Rocco Aversa, Beniamino Di Martino, Massimiliano Rak, Salvatore Venticinque, "Cloud Agency: A Mobile Agent Based Cloud System," in Proc. Int Conf. Complex, Intelligent and Software Intensive Systems, pp. 132-137, 2010.

[9] David Binkley, "Reducing the Cost of Regression Testing by Semantics Guided Test Case Selection," in IEEE Int Conf. Software Maintenance, IEEE Computer Society, Washington, pp. 251–260, October 1995.

[10] Xiaoying Bai, Wenli Dong, Wei-Tek Tsai, Yinong Chen, "WSDL-Based Automatic Test Case Generation for Web Services Testing," in Proc. Int Workshop on Service-Oriented System Engineering, 2005.

[11] Abbas Tarhini, Hacène Fouchal, Nashat Mansour, "Regression Testing Web Services-based Applications," in Proc. IEEE Int Conf. Computer Systems and Applications, pp. 163 – 170, March 8, 2006.

[12] Michael Ruth and Shengru Tu, "A Safe Regression Test Selection Technique for Web Services," in Second Int. Conf. on Internet and Web Applications and Services, 2007.

[13] Sana Azzam, Mohammed Naji Al-Kabi, Izzat Alsmadi "Web Services Testing Challenges and Approaches," ICCIT, pp. 291-296, 2012.

[14] Asir S Vedamuthu et al., "Web Services Policy 1.5 – Framework," http://www.w3.org/TR/ws-policy, April 30, 2012

[15] Corey Goldberg, "WebInject Manual," *http://www.webinject.org/manual.html*, April 30, 2012.

[16] Corey Goldberg, "Testing and Monitoring Web Services with WebInject," *http://www.webinject.org/webservices.html*, April 30, 2012.

[17] "About soapUI » Features" *http://www.soapui.org/About-SoapUI/features.html*, April 30, 2012.

[18] M. Di Penta et al., "Web Services Regression Testing" pp. 205-233, 7 May, 2007.

[19] Gregg Rothermel, Sebastian Elbaum, Alexey G, Malishevsky, Praveen Kallakuri and Xuemei Qui, "On Test Suite Composition and Cost-Effective Regression Testing", in ACM Trans. on Software Engineering and Methodology, Vol. 13, pp. 277–331, 3, July 2004,.

[20] Tamim Ahmed Khan and Reiko Heckel "A Methodology for Model-Based Regression Testing of Web Services" in Proc. IEEE Testing: Academic and Industrial Conference - Practice and Research Techniques, IEEE Computer Society, pp. 123-124, 2009.

[21] "Eucalyptus Cloud", http://www.eucalyptus.com/eucalyptus-cloud, August 8, 2012.

[22] "Github eucalyptus/wsdl," *https://github.com/eucalyptus/eucalyptus/tree/master/wsdl* , August 8, 2012

[23] Borja Sotomayor, "The Globus Toolkit 3 Programmer's Tutorial," *http://gdp.globus.org/gt3-tutorial/*, August 8, 2012.

[24] Manohar Chandwani, Hemant Mehta, Arpit Agrawal, Animesh Chaturvedi, Anand Pateria, Ajay Gupta, "Project: Implementing a Grid Environment," Devi Ahilya Univ., Institute of Engineering & Technology, Technical Report, April, 2010.