2014 IEEE 6th International Conference on Cloud Computing Technology and Science

Automated Web Service Change Management

AWSCM - A Tool

Animesh Chaturvedi
Indian Institute of Information Technology, design and Manufacturing – Jabalpur
animesh.chaturvedi88@gmail.com

***Abstract*— Automated Web Service Change Management (AWSCM) is a pioneer tool which constructs Subset WSDL based on change impact analysis of WSDL and WS code. AWSCM visualize and capture changes in the form of intermediate artifacts during impact analysis. This paper presents AWSCM modules, intermediate artifacts, components, and its applications. The paper gives insight on computation of change impact as well as mapping them to their test cases. Discussion on details of algorithm for the construction of reduce regression test suite. AWSCM can also be useful for top down development of web services using the subset operations in WSDL.**

Keywords- Web service, WSDL, regression testing, top down development, change impact analysis, SOAP and tool support.

I. INTRODUCTION

Web service (WS) is an essential part of many domains like cloud and grid computing. Changes in the requirement are an inevitable process. Change impact analysis on WS is used to automatically map indentified changes to the test cases as discussed in [8][9]. Evolution of WS is an important task that needs some automation. AWSCM does automated WS impact analysis for regression testing and top down development. We have proposed two WS regression testing approaches, namely, Operationized Regression Testing of Web Service (ORTWS) and Parameterized Regression Testing of Web Service (PRTWS) and prototyped them using AWSCM. The tool helps in selecting the relevant test cases along with test sequences/steps from the old test suite. It saves manual efforts, by automatically detecting changes in a WS and selecting the relevant test cases to construct Reduce Regression Test suite (RRTS). RRTS is used to perform efficient regression testing. Motivation of AWSCM is requirement for the improvement of automated change analysis on Services and mapping those changes to the test cases. AWSCM bridges the gap between changes on Services to its test case.

II. RELATED WORKS & TOOLS

Tool-based on WSMO (WS modeling ontology) [1] and WSDLDiff [2] are used to extract fine-grained changes from subsequent versions of WSDL. The tools like JRipple [12] and InARTS [11] provide the program organizations, impact analysis and change propagation for incremental software development. Similarly, AWSCM also comprehends the change impact analysis of WSs. Xiang Fu et al. presented a WSAT tool [14] for the formal analysis of WS, the tool works for intermediate representation, synchronizability with reliability analysis and handling of XML data manipulation.

ReSOS group [2][10] is working on the problems of re-engineering in the design and implementation of SOA based systems. Lui, Bouguettaya and Wu in [7] proposed a methodology for WS changes in replacement or addition to using top down changes in long-term composed services. Regression testing of WS is conducted for code based changes in [4][5] and model based changes in [6].

III. AWSCM TOOL

AWSCM contains two modules for regression testing, namely, ORTWS and PRTWS. ORTWS helps to find the changed operations that can be used in regression testing. PRTWS helps in finding the combinations of inputs to be exercised for regression testing. Normally, automated analysis, accessing and manipulation of WS code are performed using WSDL as interface.

AWSCM contains intermediate artifacts, packages, and components shown in fig 1. AWSCM contains five components shown in fig 1 (b). 'Code Analyzer' is used for separating method and operation, thereafter performs change impact analysis on them. 'WSDL Analyzer' is use for change impact analysis and operational analysis on WSDL. 'Diff Analyzer' uses both 'Code Analyzer' and 'WSDL Analyzer'. 'WSDL Constructor' uses the info of 'Diff Analyzer' and input WSDL to construct various Subset WSDLs. 'RRTS Constructor' is used for construction of various RRTS according to the information of change impact analysis from 'Diff Analyzer' and the old test suite.

ORTWS module has three intermediate artifacts. Changes at WSDL are captured in Difference WSDL (DWSDL) and changes at code are captured in Unit WSDL (UWSDL). Additionally, we can also capture the need of selective re-testing using another WSDL, known as Reduced WSDL (RWSDL). These intermediate forms of WSDLs combined to form Combined WSDL (CWSDL), which is used for the construction of Combined RRTS (CRRTS) as shown in figure 2. For details refer to [9] as previous version of this paper.

PRTWS module has the Parameter WSDL (PWSDL) as an intermediate artifact. Parameter WSDL is an extension of Unit WSDL for WS compositions and internal call interaction. Parameter WSDL is created with the operation whose inter-procedural flow is affected i.e. if a called method is undergone changes. PRTWS is also useful to reduce test sequences/steps in the test suite according to the code flow or user selection for the combination of a parameter scenario. PRTWS captures the flow within the operational code as shown in figure 3.

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.144

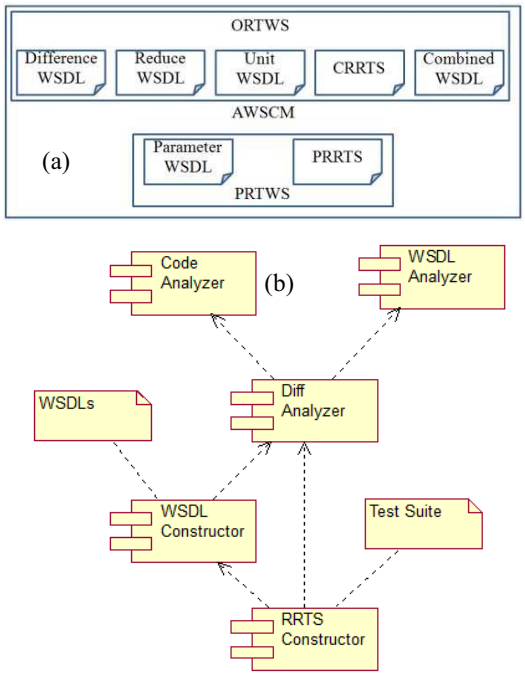

Figure 1 AWSCM (a) Intermediate Artifacts (b) Components

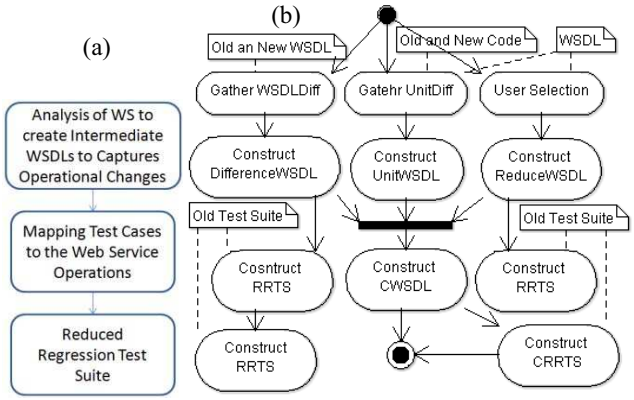

Figure 2 (a) ORTWS process (b) Activity Diagram

A. Algorithm inside 'WSDL Constructor'

All (D/U/R/P) WSDLs constructed by AWSCM are together termed as **Subset WSDL (SWSDL)**. The SWSDL further helps in accessing the **Subset Service**. Subset Service can be accessed in three ways **Difference Service, Reduce or Selective Service and Combined Service**. Change impact analysis between version 1 and 2 on a WS is used to construct the SWSDL. SWSDLs must be in proper structure and compliance with the WSDL standards. Thus, WS client can maintain communication with the WS logic via WSDL. While constructing SWSDL, special care is required for semantic, syntactic, XML tags and WSDL data correctness. That means (D/U/R/C) WSDL must have the same data of port, service, binding and schema (input/outputs of operation) as in the input WSDL. AWSCM constructs SWSDLs with correct XSD types, Message part, Operation, Port type, Binding, Port, and Service according to input WSDL and WSDL standards.

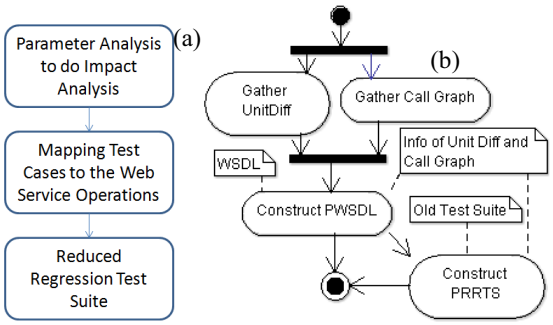

Figure 3 (a) PRTWS process (b) Activity diagram.

PWSDL are constructed with the operations that are indirectly affected from changes such that their called methods or operations have undergone changes. Called operations can also be part of other WS i.e. this is also applicable in WS composition. For the construction of 'Parameter WSDL', two inputs are required: *'newWSDL'* and *'affectedOperations'* (can be gathered after analysis of call graphs for operations which calls other operation or method affected in changes). Construction of PWSDL is as follows:

1. Gather the operations *'affectedOperations'* present in *'newWSDL'* whose inter-procedural called operation/method undergone changes.
2. Construct the various parts (Definition, XSD, Message, Port, Binding and Service) of WSDL, for the operations in step 1. Definition and Service are constant (i.e. independent of the variable operations), whereas others are dependent on the operations.
3. Combine all the parts in a proper order.

File **ConstructParameterWSDL**(File *newWSDL*, *affectedOperations*) {
String *cStartDef, cXSD, cMsg, cPort, cBinding, cService, cEndDef*
Array of String[] *affectedOperations*
[*affectedOperations* ∈ *newWSDL* | *affectedOperations* = Operation calling changed methods and operations]
cStartDef = **constructStartDefinition**(*newWSDL*)
cXSD = **constructXSD**(*newWSDL*, *newSchema*)
cMsg = **constructMessage**(*newWSDL*, *affectedOperations*)
cPort = **constructPort**(*newWSDL*, *affectedOperations*)
cBinding = **constructBinding**(*newWSDL*, *affectedOperations*)
cService = **construcService**(*newWSDL*)
cEndDef = **constructEndDef**(*newWSDL*)
ParameterWSDL = *cStartDef* + *cXSD* + *cMsg* + *cPort* + *cBinding* + *cService* + *cEndDef*
return *ParameterWSDL* }

The *DWSDL* algorithm takes two inputs: new and old WSDL. The algorithm gathers operations that have undergone WSDL changes to construct DWSDL.

File **ConstructDifferenceWSDL**(File *oldWSDL*, File newWSDL, *Difference*) {
String *cStartDef, cXSD, cMsg, cPort, cBinding, cService, cEndDef*
Array of String[] *differenceOperation, differenceSchemaOperation*
[*differenceOperation* ∈ *Difference* | *differenceOperation* = Operation for which input/output is modified or new operation is inserted in *newWSDL*]
[*differenceSchema* ∈ *Difference* | *differenceSchema* = Schema which is modified]
cStartDef = **constructStartDefinition**(*newWSDL*)
cXSD = **constructXSD**(*newWSDL*, *differnceSchema*)
cMsg = **constructMessage**(*newWSDL*, *differenceOperation*)
cPort = **constructPort**(*newWSDL*, *differenceOperation*)
cBinding = **constructBinding**(*newWSDL*, *differenceOperation*)

cService = **construcService**(*newWSDL*)
cEndDef = **constructEndDef**(*newWSDL*)
DifferenceWSDL = *cStartDef* + *cXSD* + *cMsg* + *cPort* + *cBinding* + *cService* + *cEndDef*
return *DifferenceWSDL* }

The *RWSDL* algorithm takes any WSDL as input to gather its operations and then ask the user to select operations. RWSDL is constructed with selected operations. RWSDL gives us access to the Reduce or Selective Service i.e. selective operations of a Service.

File **ConstructRedueWSDL**(File *oldWSDL*, File *newWSDL*, *Selection*) {
String *cStartDef, cXSD, cMsg, cPort, cBinding, cService, cEndDef*
Array of String[] *selectedOperation*
[*selectedOperation* ∈ *Selection* | *selectedOperation* = operations for which are selected by user]
cStartDef = **constructStartDefinition**(*newWSDL*)
cXSD = **constructXSD**(*newWSDL*, *newSchema*)
cMsg = **constructMessage**(*newWSDL*, *selectedOperation*)
cPort = **constructPort**(*newWSDL*, *selectedOperation*)
cBinding = **constructBinding**(*newWSDL*, *selectedOperation*)
cService = **construcService**(*newWSDL*)
cEndDef = **constructEndDef**(*newWSDL*)
ReduceWSDL = *cStartDef* + *cXSD* + *cMsg* + *cPort* + *cBinding* + *cService* + *cEndDef*
return *ReduceWSDL* }

The *UWSDL* algorithm takes three inputs: new, old code of operations and new WSDL. UWSDL is constructed according to the semantics of new WSDL with the operations that have gone through changes at code. PWSDL, DWSDL and UWSDL give access to the Difference Service i.e. difference or changes between the two Services.

File **ConstructUnitWSDL**(File *oldWSDL*, File *newWSDL*, *codeDifference*) {
String *cStartDef, cXSD, cMsg, cPort, cBinding, cService, cEndDef*
Array of String[] *codeDifferenceOperation*
[*codeDifferenceOperation* ∈ *codeDifference* | *codeDifferenceOperation* = Operation for which source code is modified]
cStartDef = **constructStartDefinition**(*newWSDL*)
cXSD = **constructXSD**(*newWSDL*, *newSchema*)
cMsg = **constructMessage**(*newWSDL*, *codeDifferenceOperation*)
cPort = **constructPort**(*newWSDL*, *codeDifferenceOperation*)
cBinding= **constructBinding**(*newWSDL*, *codeDifferenceOperation*)
cService = **construcService**(*newWSDL*)
cEndDef = **constructEndDef**(*newWSDL*)
UnitWSDL = *cStartDef* + *cXSD* + *cMsg* + *cPort* + *cBinding* + *cService* + *cEndDef*
return *UnitWSDL* }

CWSDL is constructed with the operations in one or more Subset WSDLs such that it contains only unique and non-redundant operations. CWSDL give access to the Combined Service i.e. combination of unique operations of Services.

File **ConstructCombineWSDL**(File *newWSDL*, *uniqueOperation*) {
String *cStartDef, cXSD, cMsg, cPort, cBinding, cService, cEndDef*
Array of String[] *uniqueOperation*
[unique*Operation* ∈ *newWSDL*Operation| *uniqueOperation* = Operation collected once if it occurred in Difference, Unit , Reduce WSDL]
cStartDef = **constructStartDefinition**(*newWSDL*)
cXSD = **constructXSD**(*newWSDL*, *newSchema*)
cMsg = **constructMessage**(*newWSDL*, *uniqueOperation*)
cPort = **constructPort**(*newWSDL*, *uniqueOperation*)
cBinding = **constructBinding**(*newWSDL*, *uniqueOperation*)
cService = **construcService**(*newWSDL*)
cEndDef = **constructEndDef**(*newWSDL*)
CombineWSDL = *cStartDef* + *cXSD* + *cMsg* + *cPort* + *cBinding* + *cService* + *cEndDef*
return *CombineWSDL* }

B. Algorithm inside 'RRTS Constructor' Component

RRTS can be constructed for the operations in a SWSDLs and the old test suite. RRTS is returned as output with test cases for only those operations of (D/U/R/C/P) WSDLs. There are two steps to construct RRTS. Firstly, gather the affected operation (using diff) as the array of *'requiredOperations'*. Secondly, pass the *'requiredOperations'* as a parameter to the function 'ReducedRegressionTestSuite'. The function uses 'requiredOperations' to gathers the required relevant regression test cases.

Suppose, WS = WS version 1, WS* = Modified WS version 2, T-old= Test Cases for the code of WS, T-new = Test Cases for the code of WS*, T* = Reduced test Cases, and *requiredOperations* = operations undergone changes. The procedure to construct RRTS is as follows

1. T* = T-old: Reduced test case initially has old test cases.
2. The declaration of operation is inserted, deleted, modified, and unmodified with their test cases.
3. If operation is deleted then delete its corresponding test cases (td).
4. If operation is inserted then add test cases (ti) template.
5. If operation is modified then add test cases (tm) with selective test sequences/steps according to changes.
6. T* = T* - tu delete all remaining unused test cases (tu) which are already executed in the testing of previous version of WS. These test cases are not required because their WS components are already tested.
7. Return the T* (reduced test case) value by applying above algorithm is evaluated to be T* = ti + tm -td – tu.

ReducedRegressionTestSuite (*requiredOperations*, T-old, T-new) {
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= deleted operation of WS
Ins= inserted operation of WS*
Mod= modified operation of WS*
td: {td ∈ T-old | td = Test cases of a deleted operation of WS}
ti: {ti ∈ T-new | ti = Test cases of a inserted operation of WS*}
tm: {tm ∈ T-new U T-old|tm = Test cases of modified operation}
tu: {tu ∈ T-old | tu = Test cases of a unmodified operation of WS}
requiredOperations: [*requiredOperations* ∈ operationInNewWSDL | operation = Operation which is required to construct RRTS]

3. if(*requiredOperations*==Del) //search for the deleted operations of WS
{ T* = T* - td //reducing test cases of deleted operations }
4. if(*requiredOperations*==Ins) // inserted operations of WS*
{ T* = T* + ti //inserting test cases of inserted operations of WS* }
5. if(*requiredOperations*==Mod) // modified operations of WS*
{ T* = T* + tm //inserting test sequences of modified operations }

6. T* = T* - tu //reducing test cases of unmodified operations
TestSuite = constructTestSuiteSemantics(T*) //{This can't be standardized!! because every testing tool have its own semantics to construct its Test Suites i.e. this step depend on testing tool semantics}

7. **Return** TestSuite //reduced regression test suite with test cases T*
}

IV. APPLICATION OF AWSCM

We find that AWSCM can be useful in both regression testing and top down development of WS. The *Regression testing of WS* was conducted with AWSCM by constructing

the RRTS with well-defined mapping of the test case with changes. The case study of ORTWS and PRTWS on real world WS is in Table 1, where Y is yes and X is no. The illustrative examples on a few WS projects demonstrate the applicability of the proposed tool for real world projects. Intermediate steps of a regression testing are important to visualize and manage the intermediate result of change impact analysis. AWSCM created the different intermediate SWSDLs to capture change impact analysis, which further helps in the construction of a RRTS. The operations of CWSDL and PWSDL are used to construct Combined RRTS (CRRTS) and Parameter RRTS (PRRTS) respectively. CRRTS and PRRTS contain all the unique test cases i.e. redundant test case were eliminated. We have removed worthless test cases to perform efficient regression testing. The PRTWS module helps in the selection of specific test sequences/steps inside a test case according to the flow change in WS code. Integration of AWSCM to the SoapUI and JMeter is in figure 4. AWSCM is just a prototype an actual realization can be done when ORTWS and PRTWS is added as feature in the standard WS testing tools. After integration GUI will be different, SWSDLs are not required to be display. The actual input required is 'New Code', 'Old Code' and 'Old WSDL' and the actual output is 'CRRTS'. CRRTS can be further reduced using test sequences/steps reducer approach.

Table 1. Case studies of ORTWS & PRTWS on Web services

<table>
<tr><th rowspan="2">Web service Project</th><th colspan="3">CWSDL for ORTWS</th><th colspan="2">PRTWS</th></tr>
<tr><th>DWSDL testing</th><th>RWSDL testing</th><th>UWSDL testing</th><th>Dynamic Black Box Testing</th><th>Static White Box Testing</th></tr>
<tr><td>Eucalyptus</td><td>Y*</td><td>Y*</td><td>Y*</td><td>X</td><td>Y*</td></tr>
<tr><td>SaaS</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr>
<tr><td>BookService</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr>
<tr><td>Amazon WS</td><td rowspan="4">Different versions of WSDL is not available</td><td>Y*</td><td rowspan="4">Code is not available</td><td>Y*</td><td rowspan="4">Code is not available</td></tr>
<tr><td>Bible WS</td><td>Y</td><td>Y</td></tr>
<tr><td>Currency Conversion WS</td><td>Y</td><td>Y</td></tr>
<tr><td>Weather WS</td><td>Y</td><td>Y</td></tr>
</table>

* *Threat to validity* because case studies is performed without the proper test data for Eucalyptus and AWS.

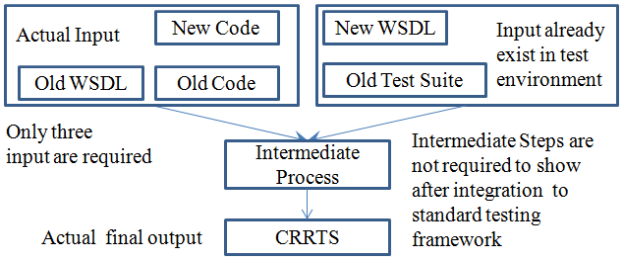

Figure 4 AWSCM integration with SoapUI and JMeter.

The *Top down development of web service* is possible with the help of subset (D/U/R/C/P) WSDLs. WS project WSDLs are given as input to AWSCM for the construction of SWSDLs. Thereafter constructed SWSDLs were given as input to the Eclipse and NetBeans, which further generate development template whose code is required to be re-engineered. We get top down development environment for black box WS with the operation in SWSDLs.

V. CONCLUSION

This paper demonstrated a pioneer tool for "Automated Web Service Change Management" (AWSCM). AWSCM had successfully created Subset WSDLs, which gives access to the Subset Service. There are three types of Subset Service, namely, Difference Service, Reduce or Selective Service, and Combined Service. Thesis [15] and tool link for detail https://sites.google.com/site/animeshchaturvedi07/research/awscm

REFERENCES

[1] El Bouhissi Houda and Malki Mimoun, "Reverse Engineering Existing Web service Application," Proc. 16th Working Conference on Reverse Engineering (WCRE, Lille Oct, 2009), pp. 279-283.

[2] R. Daniele and P. Martin, "Analyzing the Evolution of Web Services using Fine-Grained Changes," Proc. 19th IEEE International Conference on Web Services (ICWS, August 2012), pp. 392-399.

[3] B. Xiaoying, W. Dong, WT Tsai, and Y Chen, "WSDL Based Automatic Test Case Generation for Web Services Testing" Proc. Int. Work. on Service-Oriented System Engg. (SOSE, 2005), pp. 215-220.

[4] Ruth Michael and Shengru Tu, "A Safe Regression Test Selection Technique for Web Services," Proc. 2nd Int. Conf. on Internet and Web Applications and Services (ICIW, Morne June 2007).

[5] Abbas Tarhini, Hacène Fouchal, Nashat Mansour, "Regression Testing Web Services-based Applications," Proc. IEEE Int.Conf. Computer Systems and Applications (AICCSA, 2006), pp. 163 – 170.

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

[7] Liu Xumin, Athman Bouguettaya, Xiaobing Wu, and Li Zhou, "Ev-LCS: A System for the Evolution of Long-term Composed Services," IEEE Trans. on Services Computing, 99 (June 2011), pp. 1939-1374.

[8] Chaturvedi Animesh, "Reducing Cost in Regression Testing of Web Service," Proc. CSI 6th Int. Conf. on on Software Engineering on IEEE (CONSEG, Indore Sept, 2012).

[9] Chaturvedi Animesh and Gupta Atul, "A Tool Supported Approach to Perform Efficient Regression Testing of Web Service," Proc. 7th IEEE Int. Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA, Eindhoven, Sept 2013), pp. 50-55.

[10] Romano Daniele, "Analyzing the Change-proneness of Service-Oriented Systems from an Industrial Perspective," Proc. 35th Int Conf on Software Engineering (ICSE, 2013), pp. 1365-1368.

[11] P. Anjaneyulu et. al., "Selection of Regression Test Suite to Validate Software Applications upon Deployment of Upgrades," Proc. 19th Australian Conference on Software Engineering on IEEE (ASWEC, 2008), pp. 130-138

[12] Buckner J., Buchta J., Petrenko M., and Rajlich V., "JRipples: A Tool for Program Comprehension during Incremental Change," Proc. 13th Int. Workshop on Program Comprehension on IEEE, (IWPC 2005), pp. 149-152.

[13] Ren Xiaoxia, Shah Fenil, Tip Frank, Ryder Barbara G., and Chesley Ophelia. "Chianti: a Tool for Change Impact Analysis of Java Programs," ACM Sigplan Notices, 39, 10, (2004), pp. 432-448.

[14] Fu Xiang, Tevfik Bultan, and Jianwen Su, "WSAT: A Tool for Formal Analysis of Web services," Computer Aided Verification, Springer Berlin Heidelberg, 2004.

[15] Chaturvedi Animesh. "Change Impact Analysis Based Regression Testing of Web Services." *arXiv preprint arXiv:1408.1600* (2014).