Regression Testing of Service-Oriented Software

Kaushik Rana, Harshal Shah, Chetan Kapadiya

Abstract*: Regression testing is considered as a separate forms of testing attached with performance testing where tester runs old test suits after each change made to system. It will be a big problem in testing the service-oriented software (SOS), where each system component is inherently agile and changes its behavior dynamically. These agile component gives a big rise to big problem in the regression testing process with respect to complexity, time complexity and cost complexity. A service-oriented software may change in case of bug fixing, adaptation of new environment, upgrading or updating functionality in order to improve performance or it is demanded by customer. After the software is delivered to customer the service oriented software must be regressed to validate that there is no defects. We present a hierarchical regression test selection algorithm for service-oriented software, and evaluate it in service-oriented environment along with results.

***Index Terms*: Dynamic Slicing, Regression Testing, Service-Oriented Software, Testing,**

I. INTRODUCTION

Regression testing of service-oriented software composed of web service ensures that modifications due to fixing bug do not cause unexpected changes or introduce new errors or failures. As and when a bug is detected and fixed there exist potential which introduce new errors, problems or defects. Regression testing of SOS services is the selecting old test suite and retesting of an old test suite on each modification to the services or after each a bug is fixed to ensure that no new bugs have been introduced as a result of bug fixing. The main purpose of regression testing is that web services up to the point of repair have not been adversely affected by the fix. Business organizations running web services may devise guidelines to decide when to start regression testing as per their needs. This includes upgradation/updating of services, up gradation of web service description language (WSDL) or service level agreement (SLA), change in the deployment environment, and retirement of services. The actual decision regarding when to perform regression testing will be based on many factors, which includes nature of change and usage environment of services. An organization must perform regression testing in a situation where the services involve human life, environmental change or business economy change.

Fig. 1. shows that the overall regression testing process. According to Rothermel et al. [2], complete regression testing of software of 20,000 lines of code require around 2 months of continuous execution.

Revised Manuscript Received on August 05, 2019.

Kaushik K. Rana completed M.E in Computer Engineering from Dharmsinh Desai University, Nadiad, India,

Harshal Shah, Assistant Professor in Parul University, Vadodara, and Gujarat, India.

Chetan Kapadiya, Assistant Professor in Government Engineering College, Gandhinagar, and Gujarat, India.

This also necessitates development and enhancement of many existing techniques for regression testing to and be suited for service-oriented software. Thus, the problem of regression testing is formally defined as given a program P, its changed version P', and a test suite, regression testing exercise T to restore confidence on quality of P'.

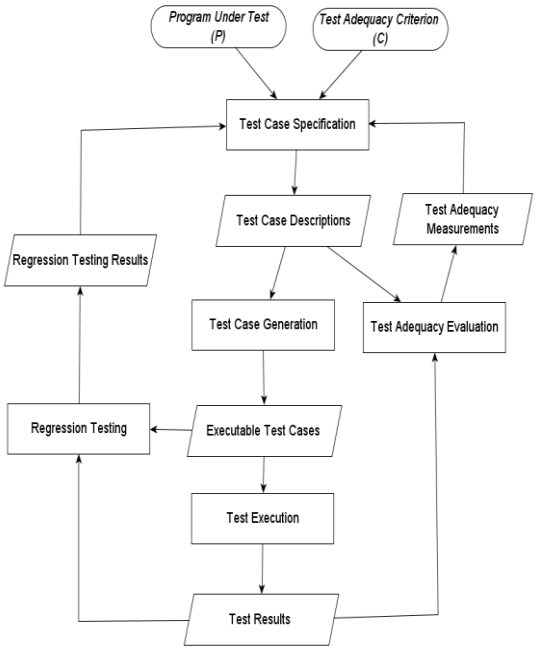

Figure 1: A software testing process model [3]

II. RELATED WORK

This section briefly presents the reported work on regression testing of SOS.

Mohanty et al. [4] planned technique for regression testing of SOA based applications. They also discuss various SOA testing challenges from the perspective of stakeholders and define whole new process for regression testing.

Bhuyan et al. [5] have carried out an extensive survey for regression testing of SOA. They also discusses the various tools for it. Ruth et al. [6] proposed technique for regression test selection for Java programs based on the approach already defined by Harrold et al. [7]. It produces Java Interclass Graph (JIG) based on compile time and run time analysis of Java programs. The JIG can calculates test cases that need to be re-executed in order to find bugs introduced if any due to change in program code. The main limitation of this technique is that it cannot be applied to the inherently distributed programs like SOS.

Gagandeep et al. [8] have

proposed a method to construct graphical web model from the analysis of web application. The web model is getting traversed to generate regression test sequences for all path coverage criteria. Ruth [9, 10] have presented a novel technique to perform safe regression test selection for both intra-enterprise and inter-enterprise web services. Bassil [11] have discussed various SOA testing challenges, existing SOA testing approaches and proposed a SOA regression testing architecture. The architecture have two parts SOA part and testing framework part. The testing framework part consists various units like test engine, test code generator, test case generator, test executer, test monitor and the database unit. Each unit works in orchestration.

Mei et al. [12] have proposed a PRT approach that addresses the dynamic binding issue. They explain their concept based on an example of Trip Handling. Hou et al. [13] have proposed two strategies for test case prioritization of regression testing of SOS. Yunus [14] have carried out hands on experiment for SOA regression testing using simple web services and SOAPSonar testing tool. Chen et al. [15] have proposed a test case prioritization technique. They construct a BPFG for their technique. Bruno et al. [16] have discussed various regression testing challenges, and have proposed an approach to provide the services with a test suite and a set of Quality of Service (QoS) constraints.

A. Web Service Example: Computing Area of Different Types of Triangle TriArea

```
                package mypackage;
                import javax.jws.WebMethod;
                import javax.jws.WebParam;
                import javax.jws.WebService;
                @WebService()
                public class TriArea {
1                   @WebMethod(operationName = "SayTriaArea")
                    public int SayTriaArea(@WebParam(name = "x")
                    int x, @WebParam(name = "y")
                    int y, @WebParam(name = "z")
                    int z)
                    {
                     int tt=4;
2                    String TType= "scalene";
3                    if((x==y)||(y==x))
4                       {TType= "isosceles";}
5                    if((x*x== y*y + z*z))
6                         {TType= "right";}
7                    if(((x==y)&& (y==z)))
8                         {TType= "equilateral";}
9                       switch(tt){
10                        case 1: TType="right";
11                                WSR wsr1 = new WSR();
12                                int port = wsr1.a1(y,z);
13                                return port;
14                        case 2: TType="equilateral";
15                                WSE wse1 = new WSE();
16                                port = wse1.a2(x);
17                                return port;
18                         default:
19                                WSD wsd1 = new WSD();
20                                port = wsd1.a3(x,y,z);
21                                return port;
                    }
```

Figure 2.1: An example web service TriArea

The WSR, WSE and WSD web service is shown in Fig. 2.2. It shows the statement numbers at left-side prefixed with WSR, WSE and WSD respectively to uniquely assign statement numbers.

```
                package mypackage;
                import javax.jws.WebMethod;
                import javax.jws.WebParam;
WSR1            import javax.jws.WebService;
                import javax.xml.ws.BindingType;
                @WebService
                @BindingType(value = "http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP
                public class WSR {
WSR2                @WebMethod(operationName = "a1")
                    public int a1(@WebParam(name = "y")
                    int y, @WebParam(name = "z")
                    int z) {
WSR3                    int t = (y *z/2);
WSR4                    return t;
                    }
                }
```

Figure 2.2 (a) A WSR web service code

```
                package mypackage;
                import javax.jws.WebMethod;
                import javax.jws.WebParam;
                import javax.jws.WebService;
WSE1            import javax.xml.ws.BindingType;
                @WebService
                @BindingType(value = "http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP
                public class WSE {
WSE2                @WebMethod(operationName = "a2")
                     public int a2(@WebParam(name = "x")
                     int x) {
WSE3                      return (int) ((x*2) * (Math.sqrt(3))/4);
                     }
                }
```

Figure 2.2 (b) A WSE web service code

```
                package mypackage;
                import javax.jws.WebMethod;
                import javax.jws.WebParam;
                import javax.jws.WebService;
WSD1            import javax.xml.ws.BindingType;
                @WebService
                @BindingType(value = "http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP
                public class WSD {
WSD2                @WebMethod(operationName = "a3")
                    public Integer a3(@WebParam(name = "x")
                    int x, @WebParam(name = "y")
                    int y, @WebParam(name = "z")
                    int z) {
WSD3                    int s=(x+y+z)/2;
WSD4                    return (int) Math.sqrt((s*(s-x)*(s-y)*(s-z)));
                    }
                }
```

Figure 2.2 (c) A WSD web service code

III. OUR PROPOSED ALGORITHM AND METHOD FOR HIERARCHICAL REGRESSION TEST SELECTION

A. Service-Oriented Software Dependence Graph (SOSDG): Our Intermediate Representation of Service-Oriented Software for Regression Testing

This section introduces a method for efficient representation of Service-Oriented Software. This representation is later used for regression testing. We name this representation as Service-Oriented Software Dependence Graph (SOSDG). Each statement of web services is represented as a node along with their number in SOSDG.

This SOSDG captures control dependencies from static analysis of web service code. It also captures data, intra-service and inter-service dependencies from run time analysis of corresponding web service execution. The inter-service dependencies may cross organizational boundaries. The web service node may belong to more than one service provider. Fig. 3.1 shows the SOSDG for Figs. 2.1, and 2.2.

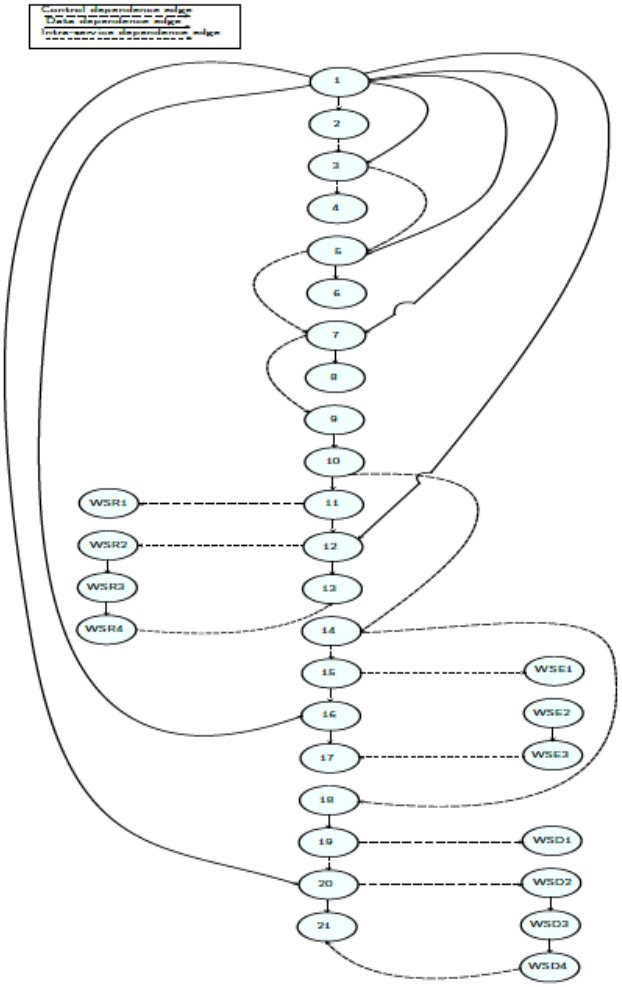

Figure 3.1 The SOSDG of the web services given in Figs. 2.1,and 2.2

B. Regression Test Selection for Web Service Algorithm

This section discusses our proposed algorithm named Regression Test Selection for Web Service (RTSWS) algorithm. We define the test cases coverage information as shown in Table 3.1. In this work we uses following sets of information:

P = (p1,p2,. . . ,pn) is the set of all the packages that are used by web services.

WSCL = (wscl1,wscl2,. . . ,wscln) is the set of all the web service classes defined in Service- Oriented Software (SOS).

WM = (wm1,wm2,. . . ,wmln) is the set of all the web service methods defined in Service- Oriented Software (SOS).

S = (s1,s2,. . . ,sn) is the set of all the statements of web services.

Table 3.1: Test cases coverage distribution for web services in Figs. 2.1, and 2.2

Test Cases	Packages (node nos)	Web Service Classes(node nos)	Web Methods (node nos)	Statements(node nos)
t1	1	1, WSE1	1, a2	1, 2, 7, 8, 9, 10, 14, 15, 16, 17, WSE1, WSE2, WSE3
t2	1	1, WSD1	1, a3	1, 2, 9, 18, 19, 20, WSD1, WSD2, WSD3, WSD4
t3	1	1, WSR1	1, a1	1, 2, 5, 6, 9, 10, 11, 12, 13, WSR1, WSR2, WSR3, WSR4
t4	1	1, WSD1	1, a3	1, 2, 9, 18, 19, 20, 21, WSD1, WSD2, WSD3, WSD4
t5	1	1, WSE1	1, a2	1, 2, 7, 8, 9, 14, 15, 16, 17, WSE1, WSE2, WSE3
t6	1	1, WSD1	1, a3	1, 2, 3, 4, 9, 18, 19, 20, 21, WSD1, WSD2, WSD3, WSD4

The RTSWS algorithm takes the SOSDG of the SOS under consideration and the test cases as its input. The RTSWS computes a forward slice w.r.t to the slicing criterion as the point of modification taken, traverses backward from each node to compute a set of affected web service statements. The slice is then decomposed into packages, web service classes, web methods and statements, respectively. Using the test case coverage analysis, the algorithm selects those test cases that affect at package level, web service class level, web method level and web service statement level, respectively. The notations used in the algorithm are:

Q - Queue that contains all the nodes reached in the forward traversal of SOSDG.

U - The set containing all the packages, web service classes, web methods and statements that are affected by the modification.

Pk - The set of packages extracted from SOSDG that are affected by the modification.

WSCL - The set of web service classes extracted from SOSDG that are affected by the modification.

WM - The set of web methods extracted from SOSDG that are affected by the modification.

S - The set of statements affected.

Now, we present our proposed RTSWS Algorithm:

Algorithm : Regression Test Selection for Web Service (RTSWS) Algorithm.

Input : Web service, Modified Web Service, Slicing Criterion, Test Suite T

Output : TR

Phase 1: Constructing Static Graphs WSCFG and WSDG

1. WSCFG Construction

(a) Node Creation

i. Define two special nodes start and stop.

ii. For each web service statement s of a web service do the the following:

A. Construct a node s.

B. Initialize the node with variables used or defined.

(b) Define control flow edges

for each web service node ni do the following

for each web service node nj do the following

Define control flow edge (ni,nj) if control flow from node ni to node nj .

2. WSDG Creation.

(a) Define control

dependence edges
for each web service predicate node ni do the following
for each web service node nj in the scope of ni do the following
Define control dependence edge (ni,nj).
(b) Define data dependence edges
for each web service node ni do the following
for each web service variable used at ni do the following
for each reaching definition nj of variable do the following
Define data dependence edge (ni,nj).
(c) Define intra-service dependence edges
for each web servicenode ni in web service Si do the following
for each web service node nj in web service Sj do the following
Define intra-service dependence edge (ni,nj) if edge is either data or control dependence edge and the state of web service Si at node ni directly depends on the execution of the node nj by web service Sj and both web services are provided within organization.
(d) Define inter-service dependence edges
for each web service node ni in service Si do the following
for each web service node nj in service Sj do the following
Define inter-service dependence edge (ni,nj) if edge is either data or control dependence edge and the state of web service Si at node ni directly depends on the execution of the node nj by web service Sj and both web services are provided by more than one service providers.

Phase 2: Compute Dynamic Slice of Web Service

1. Compute the dynamic slice of a web service using test cases in a slicing criterion such a way that each linearly independent path are covered.
(a) Let there be n number of test cases required to cover each linearly independent path in a test suite T, where T ={ t1,t2, . , tn}.
(b) Let the dynamic slice for test case ti is represented as DS (ti).
(c) Let the set of package nodes sliced by each test case ti is represented by P(ti), WSCL(ti) is the set of web service classes covered by test case ti, WM(ti) is the set of web methods covered by test case ti, and S(ti) is the set of statements covered by test case ti.

Phase 3: Compute Static Slice of Modified Web Service

(a) Initialization: Do the following for web services
i. Initialize Q, U, Pk, WSCL, WM, S to NULL.
ii. Set types ={data dependence edge, control dependence edge, interservice dependence edge, intra-service dependence edge}
iii. Add each node of WSDG that is reached by the traversal algorithm to the queue, Q.
(b) Forward Traversal
i. Traverse the WSDG using Depth First Search (DFS) algorithm in forward direction, starting from the point of modification (slicing criterion). Identify all those nodes in WSDG that are dependent on the modified statement.
ii. Add all nodes of WSDG that are reached by the DFS algorithm to the queue, Q.
(c) Backward Traversal
i. Remove the node v from Q and add it to the set U.
ii. Taking v as the starting point, traverse backward using DFS algorithm and extract all those nodes w on which node v is dependent on, such that if edge (w-> v) ε types then add all the extracted nodes w to the set U.
iii. Taking v as the starting point, traverse backward using DFS algorithm and extract all those nodes w' on which node v is dependent on, such that if edge (w'-> v) έ types then remove w' from the set U.
(d) Compute Slice
i. Find Pk = P ∩ U, if the set Pk is non-empty then we get the set of package nodes that are affected by the modification.
ii. Update U = U - Pk.
iii. Find WSCL = WSCL ∩ U, if the set WSCL is non-empty then we get the set of web service class nodes that are affected by the modification.
iv. Update U = U-WSCL.
v. Find WM = WM ∩U, if the set WM is non-empty then we get the set of affected web method nodes.
vi. Update U = U - WM.
vii. S = U.

Phase 4: Compute Regression Test Set T^R:

(e) T'= {}. Find Pt' = Pk ∩ P (ti) for each test case ti, if the set Pt' is non-empty then T'=T'U {ti}.
(f) T"= {}. Find WSCLt' = WSCL ∩ WSC (ti) for each test case ti, if the set WSCLt' is non-empty then T"=T" U {ti}.
g) T"'= {}. Find WMt' = WM ∩ WM (ti) for each test case ti, if the set WMt' is non-empty then T"'=T"'U {ti}.
(h) T""= {}. Find St' = S∩ S (ti) for each test case ti, if the set St' is non-empty then T""=T""U {ti}.
(i) TR = {T0U T" U T"'U T""}.

C. Working of the RTSWS Algorithm

We have taken an example web service shown in Figs. 2.1, and 2.2 as our case study. This program defines a web service named TriArea which calls other web services WSR, WSE and WSD. The service client invokes TriArea web service to computes the area of a triangle. The corresponding SOSDG of the program that is given as an input to the RTSWS algorithm is shown in Fig. 2.3. Suppose, the statement WSE3 of web service WSE is changed from x* 2 to x* x.

The node reached in the forward traversal from node WSE3 is 17. Then, from each of the nodes marked in the forward traversal, we traverse in the backward direction to calculate affected the marked nodes. Thus, the computed slice comprises of all the affected nodes that are finally marked by the backward traversal and are shown as light-red colored nodes in Fig. 3.2. So the set U= 1, 2, 3, 5, 7, 9, 10, 14, 15, 16, 17, WSE3.

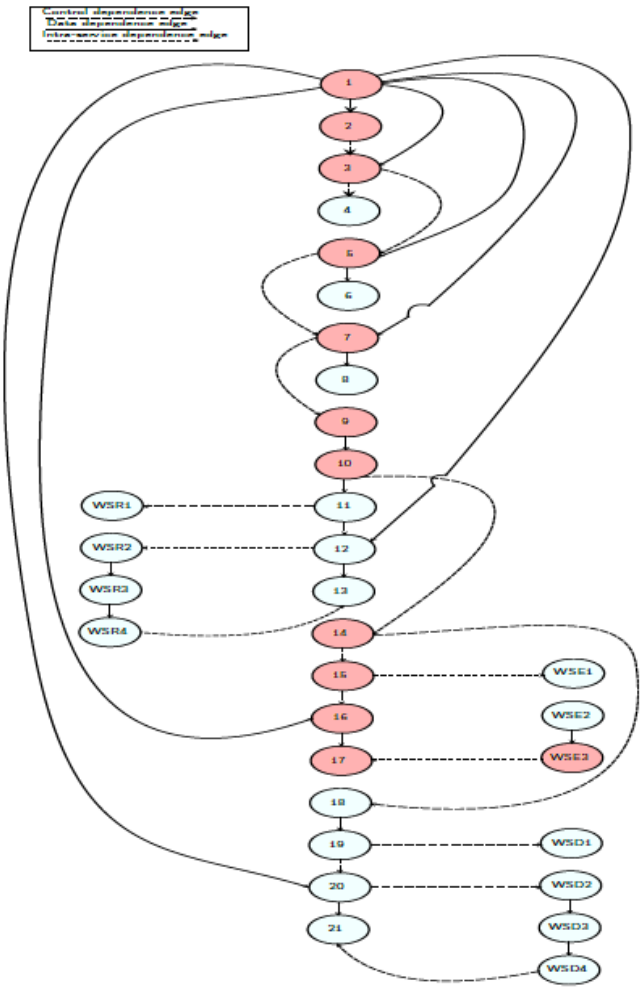

Figure 3.2 Static slice of SOSDG of the web services given inFigs. 2.1, and 2.2

The slice is then hierarchically decomposed into packages, web service classes, web methods and statements as below:

Pk = P ∩ U = {1}, and updated U= {2, 3, 5, 7, 9, 10, 14, 15, 16, 17, WSE3}.

WSCL = WSCL ∩ U = {ϕ}

WM = WM∩U = {ϕ}

S =2; 3; 5; 7; 9; 10; 14; 15; 16; 17;WSE3

Now computing regression test set as below:

InitiallyT0 =; Pt0 = Pk∩P(ti) = {1}, T'= {t1, t2, t3, t4, t5}

Next WSCLt' = WSCL∩SCL (ti) = {1}, T"= {t1, t2, t3, t4, t5}

WMt' = WM ∩ WM (ti) = {ϕ}; T''' = {t1; t2; t3; t4; t5}

Finally; St0 = S∩ S (ti) =S∩ S (t1) = {2, 7, 9, 10, 14, 15, 16, 17, WSE3}. S∩ S(t2)= {2, 9}, S∩ S(t3) = = { 2, 9}, S∩ S(t4) = { 2} and S∩ S(t5)= { 2, 7, 9, 10, 14, 15,16, 17, WSE3}

IV. CONCLUSION

We implemented our algorithm and techniques to practically verify their correctness and efficiency. We have tested slicing tool on a large number of input programs or software with several testing environment and criteria. We conclude that our regression slicer tool computed dynamic slices for SOS.

.

REFERENCES

1. Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J, "Test Case Prioritization: An Empirical Study", In the Proceedings of IEEE International Conference on Software Maintenance,pages 179-188, 1999.
2. Zhu, H., Hall, P. A., and May, J. H, "Software Unit Test Coverage and Adequacy", ACM computing surveys, pages 366-427, 1997.
3. Rajani Kanta Mohanty, Binod Kumar Pattanayak, Bhagabat Puthal, Durga Prasad Mohapatra, "A Road Map to Regression Testing of Service Oriented Architecture (SOA) Based Applications", Journal of Theoretical and Applied Information Technology, Volume 36, No 1, 15th February 2012.
4. Prachet Bhuyan, Chandra Prakash Kashyap, Durga Prasad Mohapatra, "A Survey of Regression Testing in SOA", International Journal of Computer Applications, Volume 44, No. 19 , April 2012.
5. Michael Ruth, Feng Lin and Shengru Tu, "Applying Safe Regression Test Selection Techniques to Java Web Services", International Journal of Web Services Practices, Volume 2, No.1-2, pages 1-10, 2006.
6. M. J. Harrold et al. , "Regression Test Selection for Java Software", In the Proceedings of ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA01,Tampa Bay, FL, pages 312-326, Oct. 2001.
7. Gagandeep and Dr. Jyotsna Sengupta, "Automatic Generation of Regression Test Cases for Web Components Using Domain Analysis and Modeling", International Journal of Computer Applications, Volume 11, No.12, December 2010.
8. Michael Edward Ruth, "Automating Regression Test Selection for Web Services", A Ph.D Thesis, University of New Orleans, 8-8-2007.
9. [9] Michael Edward Ruth and Shengru Tu, "Empirical Studies of a Decentralized Regression Test Selection Framework for Web Services", TAVWEB08, Seattle, WA, USA, July 21, 2008.
10. Youssef Bassil, "Distributed, Cross-Platform, and Regression Testing Architecture for Service-Oriented Architecture", Advances in Computer Science and its Applications (ACSA), Volume 1, No. 1, March 2012.
11. Lijun Mei, Ke Zhai, Bo Jiang, "Preemptive Regression Test Scheduling Strategies: A New Testing Approach to Thriving on the Volatile Service Environments", In the Proceedings of the 36th Annual International Computer Software and Applications Conference (COMPSAC2012), IEEE Computer Society, Los Alamitos, CA (2012).
12. Shan-Shan Hou, Lu Zhang, Tao Xie, Jia-Su Sun, "Quota-Constrained Test-Case Prioritization for Regression Testing of Service-Centric Systems", In ICSM, pages 257- 266, 2008.
13. Mamoon Yunus, "Intro to SOA Regression Testing: A Hands-on Approach, A White Paper, [online]. Available: www.soatesting.com, [accessed on 20/06/2019].
14. Lin Chen, Ziyuan Wang, Lei Xu, Hongmin Lu, Baowen Xu, "Test Case Prioritization for Web Service Regression Testing", 5th IEEE International Symposium on Service Oriented System Engineering, 2010.
15. Marcello Bruno, Gerardo Canfora, Massimiliano Di Penta, "Regression Testing of Web Services", [online]. Available: https://www.academia.edu/22033476/Web, [accessed on 20/06/2019].

AUTHORS PROFILE

Kaushik K. Rana completed M.E in Computer Engineering from Dharmsinh Desai University, Nadiad, India, Dr. Kauhsik K. Rana achieved Ph.D. in Computer Engineering from Gujarat Technological University in the year 2017. The author's major field of study covers Service Oriented Architecture, Software Testing and Cloud Computing. He has been working as an Assistant Professor in VGEC Chandkheda, Gujarat, India.

Harshal Shah completed M.E in Computer Engineering from Dharmsinh Desai University, Nadiad, India, Harshal Shah is pursuing Ph.D. in Computer Engineering from Gujarat Technological University. The author's major field of study covers Software Testing and Software Engineering. He has been working as an Assistant Professor in Parul University, Vadodara, and Gujarat, India.

Chetan Kapadiya completed B.E in Information Technology from Government Engineering College, Modasa, India. Chetan Kapadiya is pursuing M.E in Computer Engineering from Shankersinh Vaghela Bapu Institute of Technology, Gandhinagar. He has been working as an Assistant Professor in Government Engineering College, Gandhinagar, and Gujarat, India.