Asian Journal of Information Technology 15 (13): 2133-2146, 2016
ISSN: 1682-3915

DPHTT: A Novel Technique for Automatic Test Case Selection Designed for Regression Testing

[1]G. Parkavi and [2]D. Jeya Mala
[1]Department of Regional Center Madurai, Anna University, Chennai, India
[2]Department of MCA, Thiagarajar College of Engineering, Madurai, Tamil Nadu, India

Abstract: Regression testing which is defined as re-testing technique to test the changes has been taken from the modified or enhanced application in order to ensure the changes that do not impairment the accessible behavior of the application. Modification of the applications mainly concentrates on three type's namely binding, process and interfaces. Test cases are chosen from a test suite to accomplish the regression testing for a modified portion of an application selection and generation of the test cases are more important and also they are tough processes in regression testing. In this research, a technique has been proposed to generate the test cases automatically to test the modifications of various versions of BPEL (Business Process Execution Language) dataset. Dynamic Processing Hierarchical Test Tree (DPHTT) is formed for both new and old versions of composite services. They are changed for an application and also for the unchanged. The changes are identified by scrutinizing the control flow of both the trees formed aboveby using the BPEL dataset. The performance of the proposed technique is analyzed and the experimental results show that the proposed method performs well than the earlier techniques.

Key words: Regression testing, test case selection, BPEL, Dynamic Processing Hierarchical Test Tree (DPHTT), maintenance

INTRODUCTION

Maintenance of the applications which are designed earlier, is most expensive and important day by day. In for the purpose of maintaining the phase of an application, the application is altered according to the user requirement. The modified part of the application along with the enhancement is re-tested in order to verify that the modified parts of the application function properly. The process of re-testing an application after enhancement is called regression testing. It increases the assurance to the stability of the program that is changed by identifying errors in the program that is modified and ensuring the operation of the application functions perfectly. Though it can be carried out manually using programming methodologies, it is often taken through a software testing tool. The tool helps the environment to execute the test cases of the regression testing automatically. Regression testing is an important part of the acute programming. Figure 1 shows the flow of regression testing.

Generally, this testing method is very expensive and resource consuming process. It acts as the control measure for maintaining quality of the application. Execution of regression technique always comprises the following steps as mentioned in Fig. 2.

- All the required test cases which are against the applications are executed
- Results of the test are recorded. The results that arestored are compared with the specimen file and the differences are reported

To perform the regression testing, the test suites which are already available can be reused. Test selection

Fig. 1: Flow of regression testing

Corresponding Author: G. Parkavi, Department of Regional Center Madurai, Anna University, Chennai, India

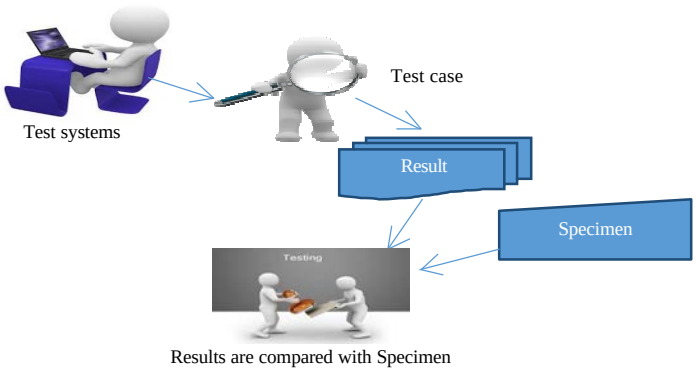

Fig. 2: Execution of regression testing

technique of regression testing helps in picking an appropriate number of test cases from a set of the test suite. All the test cases are verified for the modified program. This technique is the simplest and the safest technique for verification. Though it is a simple and safest technique, it suffers from a drawback. That is, it is practical only when the size of the test suite is small. If the test suites are larger, then few of the test suites are randomly chosen. This random selection may not have any relation with the customized program. An alternate technique for this method is selected to modify the modification revealing test cases. It executes the customized segments of the program and the segments that are affected by the modifications. The test cases which help to identify the faults in the modified segments of the program are called fault reveling test cases. Unfortunately, the process of selection and generation of test cases are always a daunting process.

In this research, the automatic generation and test cases selection are focused for the given BPEL dataset. The test cases selection can be done in four ways. Initially, the available data sets given arefully analyzed. Then, the services that are presented in the data set are selected to bind the information and to predict the constrains to execute the services. Depending on the services, a hierarchical tree is built for the process before and after modifications. Immediately after constructing the tree successfully, the defined and paved paths for the test selection are compiled. All the paths which are defined and paved for test case selection are compiled for the old and new versions. At the third stage, the paths generated for both versions of the data set are analyzed and they are compared to find the alterations in the process and binding. Besides the comparison, the message sequence flow is also compared to find the interface changes. This evaluation decides the paths that are needed to scrutinize through the test cases. After the comparison of paths of both the trees, depending on the results of comparison and mapping connection among the paths of the trees and test suites, reusable test cases of different and sub segment versions are recognized. Test cases that are generated through the above steps are experimented with various data sets. This experiment is performed to check their efficiency in finding the faults. The empirical results denote that the proposed technique has more expressive capability to find and generate the test cases. It has been designed and structured to identify three types of change types (i.e., process, binding and interface changes).

Literature review: Test case selection is the process of selecting a subset of test cases from test suite that are more suitable. The test cases are selected because their execution is relevant to the changes between the old and new versions of the file under testing. For selecting the relevant test cases, various researcheshave been carried and different approaches are proposed. The major approaches utilized to choose the more appropriate subset of the test cases are listed below:

- Integer programming approach
- Data flow approach
- Graph based approach
- Firewall approach
- Design based approach

Brief reviews of various techniques are discussed in this study.

Integer programming approach: The new way of selecting the test cases has been proposed by (Fischer (1977). Integer programming was utilized by the authors to represent the test case selection problem for testing the FORTRAN program. Similarly, a methodology was proposed related to the integer programming to find the number of testing needed to ensuremaximum possible test coverage of the test requirement (Lee and He, 1990).

Simultaneously, the researchers minimized the number of tests to be included in the test suite. In addition to that a procedure was developed based on which the test cases were determined whether it was to be eliminated or permitted. Followed by the test selection, matrix which denoted the test cases was constructed (i.e., sub-matrix from the test suite). They were used for testing the enhanced system.

In test case, the technique of reduction was handled. It reduced the test suites only when they required (i.e., on-demand) (Hao *et al*., 2012). This strategygathered the information of statistics of the capability of fault detection as an integer linear programming problem at various levels of individual statement of the difficulty of the test suite diminution. Results showed the efficient manner of reduction of different test suites. Mirarab *et al*. (2012) proposed a novel methodology of selecting the required number of test cases which already existed process. This process created an Integer Programming problem through coverage-based criteria and also constraint relaxation to find the points that were close to optimal solution. To achieve the final solution, the points which were detected were compared together with the help of retesting mechanism points were combined by voting mechanism.

Data flow approach: An incremental testing system that helps the testing during the maintenance stage is described. The testing system is designed based on the data-flow which shares the common techniques and information along with the tools. In the code of program, the Incremental tester, responses to the changes and it reutilizes the analysis and the test cases from the previous testing session. The results of the analysis were updated incrementally. These results detect the areas where retesting is required. From the updated information, the test cases which are sufficient for the modified and enhanced program are identified and they are executed to find the faults. The researchers havealso extended the above study. They incorporated the dependencies of data which exist across the procedure boundaries (Harrold and Soffa, 1989, 1988). Here, an interprocedural algorithm has been proposed to select and execute the test cases for data flow analysis. A hybrid technique has been proposed by Wong *et al*. (1997) to identify a subset of all the test cases which result in different output behavior of modified version. This technique has combined minimization; prioritization and minimization based selection of test cases. Based on the test selection to the researchers have utilized data flow analysis to approach test the spreadsheet programs. This approach has been named as What-You-See-Is-What-You-Test (WYSIWYT). WYSIWYT is a framework. Whenever the modifications are done in the cells of the spreadsheet by the user, ways in it is used to collect and update the information in the incremental fashion. This data flow analysis is possesses easy applicability in the spreadsheet.

Graph based approach: Rothermel and Harrold (1993) have proposed an algorithm to determine the test cases from the existing test suite that exhibit the fault on the new version. This algorithm has constructed a control dependence graph for all the versions of the program. It selects the test cases that are used to expose the faults in the enhanced program without the prior knowledge of modifications. Orso *et al*. (2004) the scholars have presented a new test case selection technique that is uniquely for the programs that are developed based on the java which is precise, safe and also scaled well to large systems. A tool has been proposed to implement the technique and the performance is studied on a set of test suites. The result analysis shows that the proposed technique works effectively to find the subset of test cases for the regression testing. An alternate solution is found from the article (Martins and Vieira, 2005) where the researchers have proposed a work to guide how to use the information whenever a class is modified. They have also assumed that the test suite used for the old version can be reused for the new version. However, reusing of test suites requires more effort in order to apply to new version totally. In such a situation, the subset of test suites is used and it requires additional information besides the source code. The study presented by Li *et al*. (2012) and Ii *et al*. (2002) would help to collect such extra information by constructing directed graph from the activity diagram which is used to capture the behavior of reusable classes. Such graphs are named as Behavioral Control Flow Graph (BCFG). The main strength of the Graph based approach is its generic applicability.

Graph based techniques can also be applied to test the web services. But, it is a difficult challenge to implement due to the distributed nature of the web service. To overcome this challenge, JIG-based methodology has been proposed by Lin *et al*. (2006). It is applied after web services are transferred to a single JVM local application. Apart from this, a framework is designed by Binkley (1997) to carry out the regression test case selection in an efficient manner.

Firewall approach: In this approach (Leung and White, 1990), the scholars have built a firewall around the sections or modules of the program where retesting is needed. Here, for any modified section of the program, the researchers have chosen the test cases that integrate modules and they are modified. This firewall approach is

also applied in object-oriented programming (White and Robinson, 2004) and in GUIs. They are presented in (White *et al*., 2003). Zheng *et al*. (2007) have successfully implemented the firewall technique based on an information extracted from the deployed binary code. In addition firewall technique has been applied to a large-scale banking system in Skoglund and Runeson.

Design based approach: Briand *et al*. (2009) have presented a technique and a tool to support test case selection for regression testing from the test suites based on change analysis in Object-Oriented design. An assumption is made that the UML is designed and a formal mapping is proposed between design changes and test cases of the regression testing. Which is classified into three categories.

Other approaches: A new event-driven paradigm is presented by Kumar and Goel (2012) where the test cases are automatically compared. This event-driven technique is based on the creation of dependencies among the event as a graph. The graph is drawn for all the versions. Then it is converted to the trees. The trees are compared to detect the potentially affected nodes that enable the test case selection. Based on the hierarchical slicing method, a new technique has been proposed by Tao *et al*. (2010). This method has improved the precision of regression test selection.

MATERIALS AND METHODS

In this study, a technique, that drives the test case selection for regression testing is presented. Three major steps are followed after selecting the services to select the test cases. The tree steps are as follows:

- Tree construction: it is the process of constructing the hierarchical tree namely Dynamic Processing Hierarchical Test Tree (DPHTT) for the different versions of BPEL data set (i.e., versions represent the data set before and after enhancement of the application's composite services)
- Path computation: from the tree constructed in step 1, the path which exists among the sequence of message is detected and it is computed to find the modifications
- Path comparison: this step is proceeded to find the modifications in the process, interface and binding

From the comparison of path between the old and new versions of the BPEL dataset, the modification can be found. Once the modifications, the test cases, that are required to test the process, are identified. Figure 3 presents the overall structure of the proposed study. The algorithm 1 shows the procedure to generate the test cases for the detection of changes in the new version. The detailed explanations of the above steps are given in the following sub sections.

Tree construction: From the BPEL data set, a DPHTT which represents all the processes such as partner likes services, variable, etc., is constructed. The DPHTT which is built is utilized to explain the characteristic behavior of the composite services present in the BPEL data set. The DPHTTcomprises a root node. It represents the process name of the BPEL file. For the root node, child node are DPHTTcomprises a root node. It represents the process name of the BPEL file. For the root node, child node are framed. They provide the message of the sub-tags of the process tag. All the sub-tags of the process tag will present the degree one. These child nodes are denoted as categories and by using those the comparisons are done.

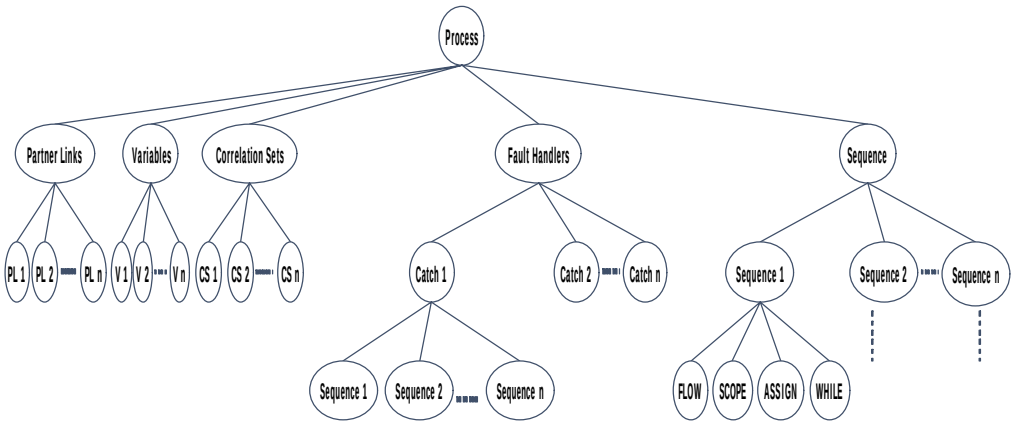

Fig. 3: General construction of DPHTT

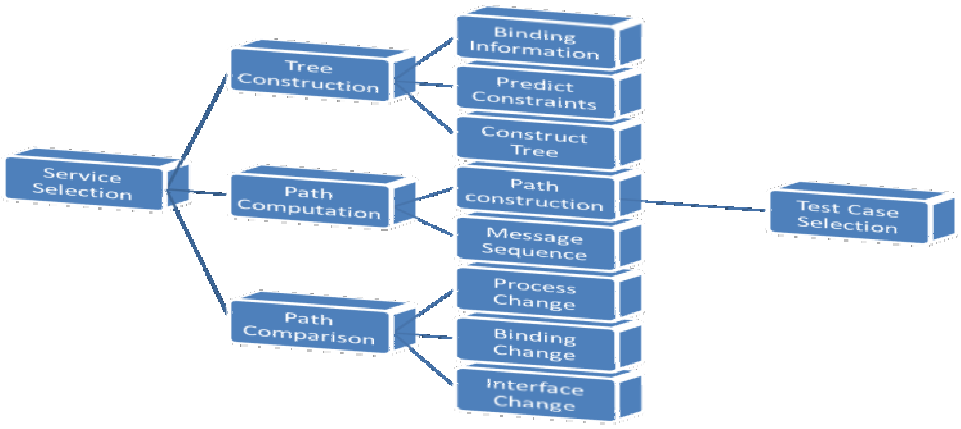

Fig. 4: Overall flow of proposed method

```
- <process name="AtmFrontEnd" targetNamespace="http://jbpm.org/examples/atm"
xmlns:tns="http://jbpm.org/examples/atm" xmlns:atm="http://jbpm.org/examples/atm"
xmlns:tic="http://jbpm.org/examples/ticket" xmlns:acc="http://jbpm.org/examples/account"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
process/">
- <partnerLinks>
  <partnerLink name="atm" partnerLinkType="tns:Atm-Front" myRole="FrontEnd" />
  <partnerLink name="ticket" partnerLinkType="tns:Front-Ticket" partnerRole="TicketIssuer" />
  <partnerLink name="account" partnerLinkType="tns:Front-Account" partnerRole="AccountSystem" />
  </partnerLinks>
- <variables>
  <variable name="ticketMsg" messageType="tic:ticketMessage" />
  <variable name="connected" type="xsd:boolean" />
  <variable name="logged" type="xsd:boolean" />
  </variables>
- <correlationSets>
  <correlationSet name="atmInteraction" properties="tns:ticketId" />
  </correlationSets>
- <sequence name="MainSeq">
+ <scope name="TicketCreationUnit">

+ <assign name="InitializeStatus">

+ <scope name="ConnectionUnit">
  </sequence>
  </process>
```

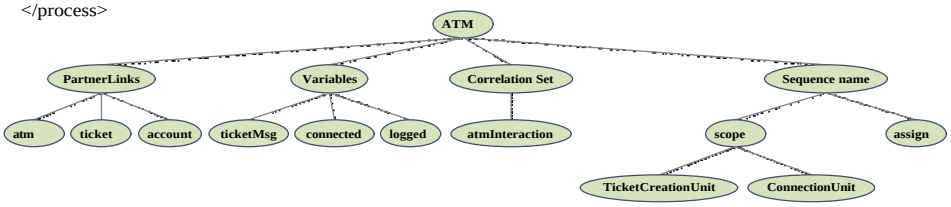

Fig. 5: Example of ATM BPEL file (version 1.0)

Similarly, the sub-tags of the BPEL file are arranged to form a DPHTT. Figure 4 represents the general DPHTT. The tree must be structured for both the old and new versions of the BPEL files.

With this process, the information binding and the constraints are determined. They are added to the DPHTT and they are helpful for the path computation and comparison process. A sample DPHTT tree construction for BPEL file of ATM for version 1.0 is presented in Fig. 5 along with the source code.

Algorithm 1; Path computation:

1 begin
2 Input path[count]: current Tree path to be processed;
3 Input em: current Tree element to be processed;
4 Output path[count]: all Tree s to be generated;
5 Variable op[mcount]: all Tree out-process s generated;
6 Process(path[count], em)
7 //termination condition of recursion
8 if em==null then
9 return
10 end if
11 if em.category==SN then
12 //SN- Service Node – Created for partner service that is defined by partner links
13 path[count]=path[count]? {em};

```
14 for each emi ? em.oldurce or em.target do
15 if emi.category == ME then
16 //ME – Message Edge for message exchange between BPEL and WSDL interface
17 mcount ++;
18 create a new TREE out-process op[mcount];
19 end if
20 //create out-process from ME
21 CreateOP(op[mcount], emi);
22 path[count]=path[count]? op[mcount];
23 end for
24 for each emi ? em.target do
25 if emi.category!=ME then
26 Process(path[count], emi);
27 end if
28 end for
29 else if em.category ==IN then
30 //IN – Interaction Node – created for basic activities which interact with the partner service.
31 path[count]=path[count]? {em}
32 for each emi ? em.oldurce do
33 if emi /? path[count]
34 pTemp=path[count];
35 if i > 1 then
36 count++;
37 create a new TREE path[count];
38 Path[count]=pTemp;
39 end if
40 end if
41 path[count]=path[count]? {emi};
42 Process(path[count], emi)
43 end for
44 else if em.category==default then
45 path[count]=path[count]? {em}
46 Process(path[count], em.target);
47 end if
48 end
```

Algorithm 2; path comparison:

```
1 begin
2 // comparioldn is to find the affected by process change and binding change
3 // Process[i] – Process version 1, Process [i+1] – Process version 2
4 Input Process[i], Process [i+1]: set of s to be compared;
5 Input No[i], No [i+1]: set of elements in Process[i] and Process[i+1];
6 Output Process [i+1] old: set of old paths;
7 Output Process [i+1] new: set of new paths;
8 Comparioldn (Process[i], Process[i+1], No[i], No[i+1])
9 Let Nall =No[i] ?No [I+1]
10 for each element e? Null do
11 if e /? No[i] ? e? No[i+1] then
12 No [i+1] add=No[i+1]add? {e}
13 if n.category==EN) then
14 //EN – Exclusive Node – created for the activities that contain conditional //behavior.
15 No [i+1] _add=No [i+1] _add? {e}
16 end if
17 if n.category==SN && n.category==IN ) then
18 No [i+1] _add=No [i+1] _add? {e}
19 end if
20 else if e? No[i]? e /? No[i+1] then
21 No[i] Del=No[i] del? {e}
22 end if
23 end for
24 for each element n_add? No[i+1]_add do
25 for each element of Process[i+1]? Process[i+1] do
26 if n_add? Process[i+1] then
27 Process [i+1] new =Process [i+1] new ? {Process [I+1]}
28 end if
29 end for
30 end for
31 Process [i+1] old =Process [i+1] s \Process[i+1]new
32 for each Process[i]? Process[i] do
33 for each No[i]del? No[i]del do
34 if No[i]del? Process[i] then
35 NProcess[i] Del=NProcess[i] del? {No[i]del}
36 end if
37 end for
38 if Process[i+1]=(Process[i]-NProcess[i]del )? Process[i+1]
39 Process [i+1] new =Process [i+1] new ? {Process [I+1]}
40 end if
41 end for
42 Return Process [i+1] old, Process [i+1] new
43 end
```

Path computation: Path computation is the process which calculates the number of process operations, message operations, invoke operations and conditions. From the DPHTT tree constructed in step one is used for the computation. From the root node, i.e., from the process node, the traversal starts along all the nodes that are present at degree one that is the child of the root node and through which all the categories presented will be found out. The partner link istraverses and the total number of partner links is equal to the number of children the partner link node contains. To help tree traversal, a service node is created. It keeps track the nodes that are still to be traversed. Similarly, the variable, correlation sets and all other attributes present in the DPHTT are computed. Operations and the conditions can be calculated with the help of traversing sequence node.The corresponding child nodes which provide the total number of conditions, process, invoke operations and message sequence are available, present in BPEL file. If there is no child node for the root node, it is clear that there is no category for that process. The path calculation process is done for both the old and new versions of BPEL file. If Fig. 5 is considered in which the path computation is carried out as follows:

Path computation:

- Partner links: 3
- Variables: 3
- Correlation: 1
- Sequence: 2 (this contains further sub tags that are not shown in Fig. 5 on expansion of the source code DPHTT tree also grows)

Message sequence: To find the modified portion of the BPEL file, path comparison is utilized. It detects the process, binding and interface change. Process change can be identified by using the BPFL file it self. At the

```
-<wsdl name="ATM WSDL INTERFACE">
-<definitions targetNamespace="DPHTTp://jbpm.org/examples/ticket"
xmlns:tns="DPHTTp://jbpm.org/examples/ticket"
xmlns:xsd="DPHTTp://www.w3.org/2001/XMLSchema"
xmlns="DPHTTp://schemas.xmlsoap.org/wsdl/">
+<message name="ticketRequest">
+<message name="ticketMessage">
+<portType name="TicketIssuer">
 </definitions>
-<definitions targetNamespace="DPHTTp://jbpm.org/examples/account"
xmlns:tns="DPHTTp://jbpm.org/examples/account" xmlns:xsd="DPHTT
p://www.w3.org/2001/XMLSchema" xmlns="DPHTTp://schemas.xmlsoap
org/wsdl/">
+<types>
+<message name="customerMessage">
+<message name="accessMessage">
+<message name="balanceMessage">
+<message name="accountOperation">
+<portType name="AccountSystem">
 </definitions>
-<definitions targetNamespace="DPHTTp://jbpm.org/examples/atm"
xmlns:tns="DPHTTp://jbpm.org/examples/atm"
xmlns:tic="DPHTTp://jbpm.org/examples/ticket"
xmlns:acc="DPHTTp://jbpm.org/examples/account"
xmlns:xsd="DPHTTp://www.w3.org/2001/XMLSchema"
xmlns="DPHTTp://schemas.xmlsoap.org/wsdl/">
        <import namespace="DPHTTp://jbpm.org/examples/ticket"
location="ticket.wsdl" />
      <import namespace="DPHTTp://jbpm.org/examples/account"
location="account.wsdl" />
+<types>
 <message name="connectRequest" />
+<message name="logOnRequest">
 <message name="logOnResponse" />
+<message name="statusResponse">
+<message name="balanceChange">
+<message name="unauthorizedAccess">
+<message name="insufficientFunds">
+<portType name="FrontEnd">
 </definitions>
 </wsdl>
```

Fig. 6: Message sequences in ATM WSDL file (version 1.0)

same time, BPEL file is not sufficient to detect the binding and the interface change. In BPEL files processes that interact with the partner links can be found and whose exposed interfaces are found in the corresponding WSDL file. To detect the interactive behavior, the order in which the input and output messages are exchanged is used depending on the input and output variables of BPEL file. Therefore, it is mandatory to invoke the WSDL file which corresponds to the BPEL file and it is taken in the present work. Likewise, the message sequence is identified in the corresponding WSDL file. Figuer 6 denotes the portion of the ATM WSDL that corresponds to the ATM BPEL file. It is chosen and it contains the message sequences. These message sequences

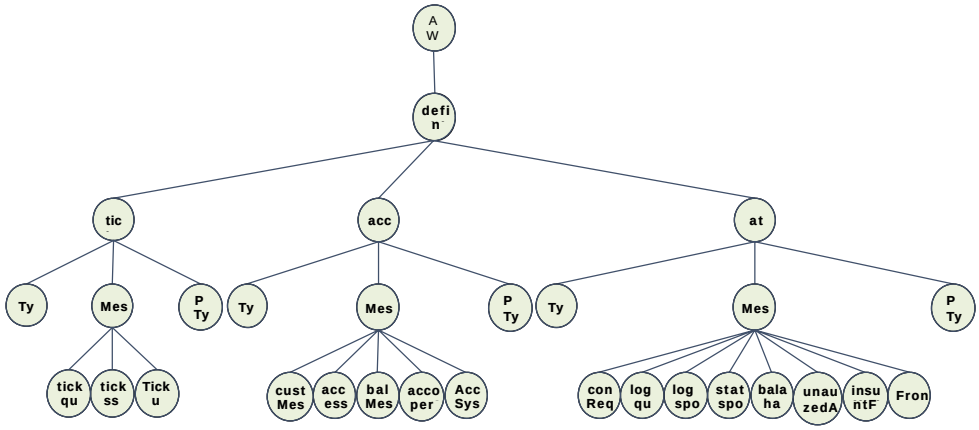

Fig. 7: The DPHTT tree representation of message sequence

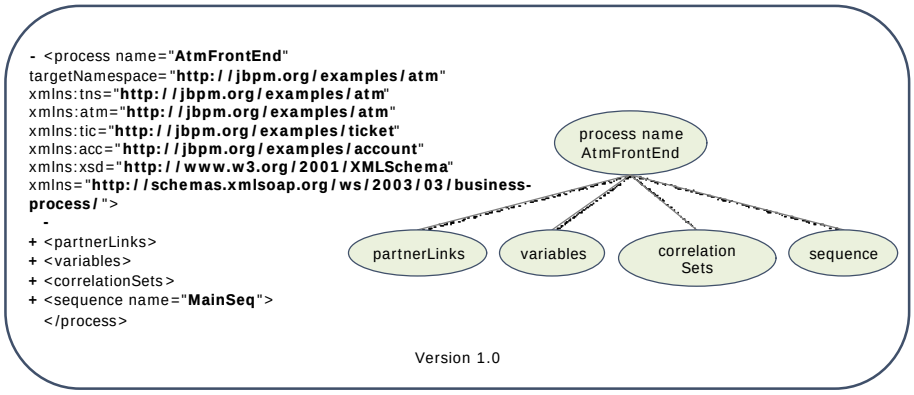

Fig. 8: BPEL file of ATM (version 1.0)

correspond to the function call in WSDL files. These message sequences are also tracked and drawn in the DPHTT tree. The subtree of DPHTT which contains only the message sequence is shown in Fig. 7.

Path comparison

Process change: This type of change shows the changes in the internal structure of the BPEL file. Internal changes of the BPEL file happen as a result of insertion or change of services depending on the user's functional need. Changes which occur in the execution sequence and activities are treated as process change. Changes in the process can be found with the assistance of the size of the BPEL file. The process change is expressed either by replacement or by increasing the size of the process.

Binding change: Modifications will after the addresses of the partner links in the functions of the BPEL file. This change will become the reason for the variations while choosing different candidate service and it will replace the original one that is not available now. Hence, these types of changes can be deleted by using the message sequence change.

Interface change: Similar to the binding change, the interface change can be found using the changes in message sequence and it is determined through the DPHTT of WSDL file. This type of change comprises the composite and partner service interface changes. These interfaces contain the explanations of the messages, ports, variables and operations. In most of the cases, the interface modifications are considered to improve the ability of the program and the readability of the WSDL files. These modifications of the partner link will indicate the service integrator to change the corresponding interface.

For example, the ATM examples with changes are considered they exist between two different versions of BPEL files along with their corresponding interface changes in the WSDL file. Figure 8 and 9 shows the

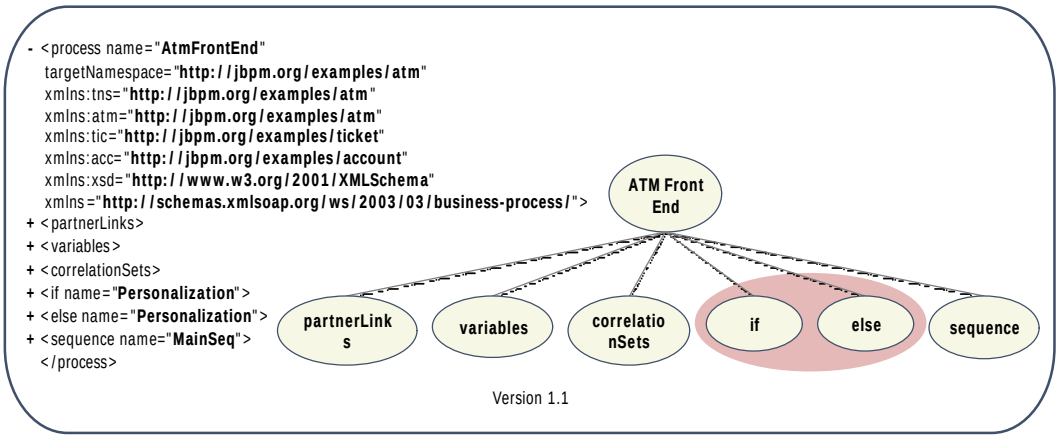

Fig. 9: BPEL file of ATM (version 1.1)

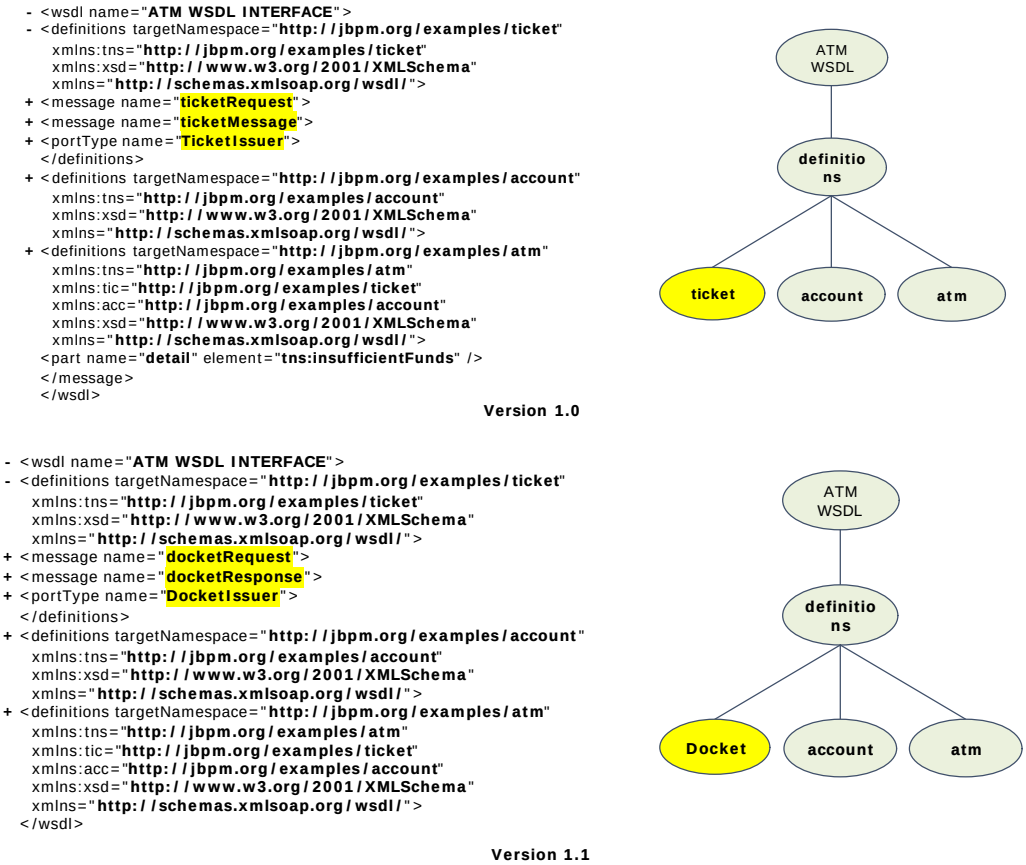

Fig. 10: Modifications in WSDL file of ATM

difference in the two BPEL files of ATM. The encircled portion in Fig. 8 expresses the enhancement that is taken in version 1.1. Similar changes can be carried out in all the programs through expanding the sub tags of the BPEL file and thereby DPHTT tree also grows. The comparison can be found through traversing the tree along the path of the sub trees. Here, only degrees zero and one of DPHTT are presented. Similarly has to be expanded the tree. In similar manner, the WSDL files can be compared to detect the interface and binding changes. Figure 10 denotes the modifications in the WSDL file.

Test case selection: The changes which are identified in the new versions of the BPEL files have to be examined to

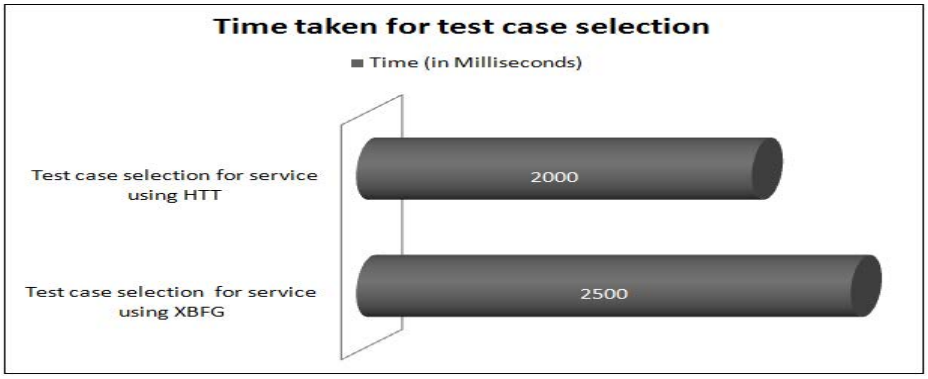

Fig. 11: Time taken for tests case selection

make sure that they do not affect the fundamental functionalities. The test can be performed by using the test cases. The selection of test cases is the challenging task for testing the modified part. For choosing the test cases, the modified path in the DPHTT tree is computed. The modified path consists of two paths namely:

- Old path
- New path

Old pathrefers the test cases that alreadyexist for the baseline version andit is selected in order to re-run. In the case of new path, new test cases should be generated. Both the paths are analyzed to decrease the number of test cases required to test the modified part of the program. Since, the test cases can be applied for the baseline version, they can be adopted for the new path as well. In addition to the common test cases, new test cases can also be added for regression testing.

A simple case study: For ATM, all DPHTT paths and corresponding message sequences of v1.0 and v2.0 are built. Three types of changes which are presented in the enhanced version are detected and manipulated. From section 3, the process change, binding change and interface changes are found. The process changes are detected among the two versions v1.0 and v2.0 of the ATM BPEL files, when the number of elements present in the version v2.0 is either greater than or not equal to the version v1.0. The process changes can be expressed mathematically as follows:

$$ps[1.0] = e1 \cup e2 \cup e3 \cup e4$$
$$ps[2.0] = e1 \cup en2 \cup e3 \cup e4 \cup e5 \cup e6$$
$$\text{Condition: } ps[1.0] \neq ps[2.0] \| ps[1.0] < ps[2.0]$$

From Fig. 8 and 9, we can find the number of elements can be found for version v1.0 and v2.0, respectively as follows:

$$ps[1.0] = \text{partner link} \cup \text{Variables} \cup \text{Correlation set} \cup \text{sequence}$$
$$ps[2.0] = \text{partner link} \cup \text{Variables} \cup \text{Correlation set} \cup \text{if} \cup \text{else} \cup \text{sequence}$$

From the above equation, it is evident that the number of element present in the ps (1.0) is lesser than the ps (2.0). Therefore, it shows that ps (1.0) = ps (2.0)as well as ps (1.0)<ps (2.0). This implicitly expresses the existence of process change in the modified version of 1.0 (ATM's BPEL file), i.e. version 2.0. The elements that are taken for example are only the tags and are immediate children of the root node. Likewise, the binding change occurs, when changes in the services that are bind with the process change, exist. Mathematically the binding chance can be explained as below:

$$bs[1.0] \supset \{b1, b2, b3, b4, b5\}$$
$$bs[2.0] \supset \{nb1, nb2, nb3, nb4, nb5\}$$
$$\text{Condition: } bs[1.0] \neq bs[2.0]$$

Figure 11 shows the binding changes that occur in both the versions of ATM BPEL file. It is evident from Fig. 11 that the binding changes occur with the modifications in the service name. The service name of ticket in v1.0 is modified as docket in v2.0. The example presented in Fig. 11 is a part of BPEL file and it is not for the entire process:

$$bs[1.0] \supset \{atm, ticket, account\}$$
$$bs[2.0] \supset \{atm, ticket, account\}$$
$$\Rightarrow bs[1.0] \neq bs[2.0]$$

Similar to the process and binding change, interface change can also be detected. To find this, the messages

version 1

```
<partnerLinks>
- <!-- relationship with the ATM
 -->
 <partnerLink name="atm" partnerLinkType="tns:Atm-Front" myRole="FrontEnd" />
- <!-- relationship with the ticket issuer
 -->
 <partnerLink name="ticket" partnerLinkType="tns:Front-Ticket" partnerRole="TicketIssuer"
/>
- <!-- relationship with the account system
 -->
 <partnerLink name="account" partnerLinkType="tns:Front-Account"
partnerRole="AccountSystem" />
 </partnerLinks>
```

version 2

```
<partnerLinks>
- <!-- relationship with the ATM
 -->
 <partnerLink name="atm" partnerLinkType="tns:Atm-Front" myRole="FrontEnd" />
- <!-- relationship with the ticket issuer
 -->
 <partnerLink name="docket" partnerLinkType="tns:Front-Ticket" partnerRole="DocketIssuer"
/>
- <!-- relationship with the account system
 -->
 <partnerLink name="account" partnerLinkType="tns:Front-Account"
partnerRole="AccountSystem" />
 </partnerLinks>
```

Fig. 12: Binding change

which are received and sent to the WSDL process are considered. The interface change can be mathematically represented as follows:

$$is[1.0] \supset \{m1,m2,m3,m4,m5\}$$
$$is[2.0] \supset \{nm1,nm2,nm3,nm4,nm5\}$$
$$\text{Condition: } si[1.0] \neq is[2.0]$$

Consider Fig. 12 and 13. It denotes that the message used in v1.0 is changed in v2.0. That is shown precisely as below:

$$is[1.0] \supset \{\text{ticket Request}, \text{ticketMassage}\}$$
$$is[2.0] \supset \{\text{docket request}, \text{docket response}\}$$
$$\Rightarrow si[1.0] \neq is[2.0]$$

The name of the changes and its location that are presented in the enhanced version are presented in the Table 1. Path details of DPHTT are show in Table 2. The DPHTT path is shown in Table 2 for two different versions of ATM files. The numbers present in the table represent the node number to which an element of the files is presented in the DPHTT. The highlighted part shows the nodes path at which the changes occur.

Table 1: Changes and its location

Changes type	Location	Enhancement result
Process chang	BPEL	Process chang occurs
Binding change	BPEL	Binding change occurs
Interface change	WSDL	Interface change occurs

Table 2: DPHTT path

Version	HTT path
P1 (1.0)	3, 5, 7, 11, 2, 4, 6, 10, 17, 23, 25, 12, 13, 14, 30, 27, 34, 35, 28, 29, 40, 22, 19, 31, 32, 37, 41, 45, 46, 2, 1, 9, 15, 47, 22, 8, 16, 43, 44, 38
P2 (1.0)	3, 5, 7, 11, 2, 4, 6, 10, 17, 23, 25, 12, 13, 14, 30, 27, 34, 35, 28, 29, 40, 22, 19, 31, 32, 37, 41, 45, 46, 2, 1, 9, 15, 47, 22, 8, 16, 43, 44, 38
P1 (2.0)	3, 5, 7, 11, 2, 4, 6, 54, 49, 52, 57, 60, 51, 0, 17, 23, 25, 12, 13, 14, 30, 27, 34, 35, 28, 29, 40, 22, 19, 31, 32, 37, 41, 45, 46, 2, 1, 9, 15, 47, 22, 8, 16, 43, 52, 64, 53, 55, 68, 66, 59, 44, 38
P2 (2.0)	3, 5, 7, 11, 2, 4, 6, 54, 49, 52, 51, 60, 57, 10, 17, 23, 25, 12, 13, 30, 14, 34, 27, 35, 28, 29, 40, 22, 19, 31, 32, 37, 41, 45, 46, 2, 1, 47, 22, 22, 9, 15, 8, 16, 43, 52, 64, 53, 55, 68, 66, 59, 44, 38,
P3 (2.0)	3, 5, 7, 11, 2, 4, 6, 54, 49, 52, 51, 60, 57, 56, 10, 17, 23, 25, 12, 13, 30, 14, 34, 27, 35, 28, 29, 40, 22, 19, 31, 32, 37, 41, 45, 46, 21, 47, 22, 9, 15, 8, 16, 43, 52, 64, 53, 55, 59, 68, 66, 44, 38

version 1

```
<message name="ticketRequest">
  <documentation>ticket creation request</documentation>
 </message>

 <message name="ticketMessage">
  <documentation>ticket number wrapper</documentation>
  <part name="ticketNo" type="xsd:int" />
 </message>
 <portType name="TicketIssuer">
```

version 2

```
<message name="docketRequest">
  <documentation>ticket creation request</documentation>
 </message>

 <message name="docketResponse">
  <documentation>ticket number wrapper</documentation>
  <part name="ticketNo" type="xsd:int" />
 </message>

 <portType name="DocketIssuer">
```

Fig. 13: Interface change

RESULTS AND DISCUSSION

To verify the proposed study, two different versions of BPEL files of ATM are taken. For both the versions of BPEL file, DPHTT tree is builtand from which the path is computed and compared to detect the modifications such as process, interface and binding. Once the modifications are found, the test cases which are suitable to test the changed or enhanced portion of the application are chosen in order to provide the confidence that the modified portion does not affect the other parts of the application. Efficiency of the proposed work is analyzed in determining the test cases is by using processing time, time taken for selecting the test cases and time taken to detect the faults using the selected test cases as major performance metric. The proposed method is compared with the existing graph based on technique name by XBFG. The analysis and their results are expressed in the following sub-section.

Time taken for test case selection: Test case selection requires considerable amount of time. Therefore, this metric acts as a major role in deciding the time factor for the algorithms. Figure 12 and 13 shows that the time taken for selecting the test cases through the graph based algorithm is more than the tree based proposed technique. Time measurements are carried out in (ms). The proposed tree based DPHTT technique DPHTT requires 2000 ms whereas graph based XBFG technique need 2500 ms.

Time taken for fault detection in test cases: Another key metric in the analysis of the test case selection is the time taken to detect the faults that have occurred because of the modified portion of the application. If the algorithm detects the faults that occur due to the algorithm faster it implicitly improves the running time of the entire algorithm. From Fig. 14, it is understand that the proposed DPHTTtechnique consumes lesser time than the graph based technique XBFG.

The proposed method consumes 1700 ms in order to fix the faults whereas the existing graph based technique needs 3000 ms. The processing time necessary for both test case selection and fault detection is lesser for the proposed method and thereby, the proposed DPHTT technique implicitly reduces the overall time needed to execute the process.

Processing memory: Memory usage is the primary metric that determines the required memory to execute the techniques. Figure 14 expresses the utilization of the main memory during the execution of both the proposed and existing techniques. The memory usage is expressed in bytes. Figure 15 portrays that DPHTT requires lesser main memory than the existing method.

Other metrics: Number of parameters must be decided earlier for graph, i.e., the number of sub tags should be known earlier before the graph is drawn for the BPEL files. Initially, a graph should be constructed for any additional

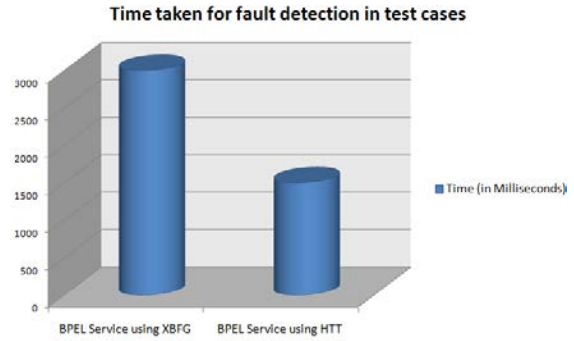

Fig. 14: Time taken for fault detection in test cases

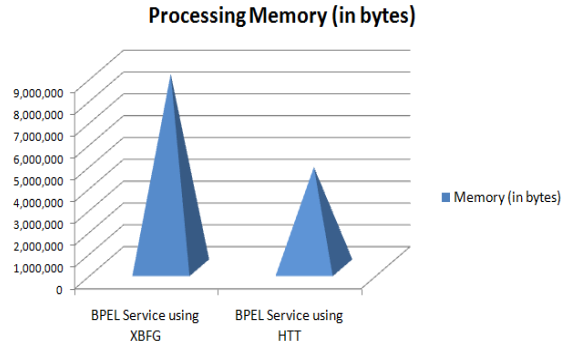

Fig. 15: Memory usage

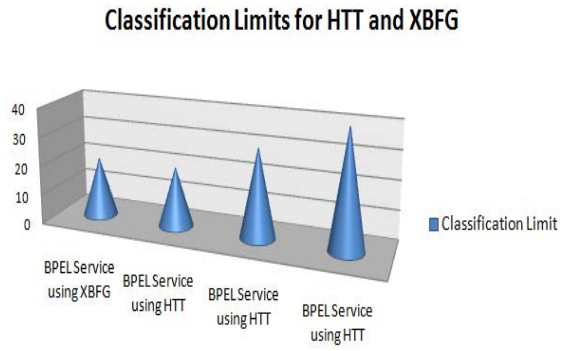

Fig. 16: Applicable conditions

sub tags. Therefore, for all different versions, graphs should be drawn separately. But, it is not in the case of tree based construction. Any modifications can be added without building it from preliminary level. Figure 16 shows that a graph based technique cannot be modified, once it is drawn for a version with 20 conditions. It should be constructed from the preparatory stage for next version with 30 and 40 conditions correspondingly. Whereas the proposed DPHTT based technique can be modified for the new versions of the BPEL file and it can use the trees that are drawn for earlier versions.

CONCLUSION

Modifications or enhancements in the services of an application make an immense challenge to testing and maintaining the application. In the presented research a DPHTT-based regression testing technique is proposed. It helps in detecting the influence caused by binding, process and interface change. Initially, DPHTT tree is constructed for the different versions of the BPEL files that are to be tested. From the DPHTT tree, paths are generated and computed. Computed paths are compared in order to detect the modifications among both versions. Depending on the comparison result, two different types of test cases are generated; first part from the baseline version and second part of test case is generated newly. These are used to detect the modified portion of the application. Experimental results show that DPHTT works well than the existing method named XBFG. This proposed work analyses only how to retest the modified or enhanced portions of the BPEL files and it is considered as one of the many service composition languages. If the services are composed based on other composition languages such as WS-CDL, OWL-S how to deal with the maintenance and regression testing is in the future study.

REFERENCES

Binkley, D., 1997. Semantics guided regression test cost reduction. IEEE Trans. Software Eng., 23: 498-516.

Briand, L.C., Y. Labiche and S. He, 2009. Automating regression test selection based on UML designs. Inf. Software Technol., 51: 16-30.

Fischer, K.F., 1977. A test case selection method for the validation of software maintenance modifications. Proc. COMPSAC., 77: 421-426.

Hao, D., L. Zhang, X. Wu, H. Mei and G. Rothermel, 2012. On-demand test suite reduction. Proceedings of the 34th International IEEE. Conference on Software Engineering, June 2-9, 2012, IEEE, Piscataway, USA., ISBN: 978-1-4673-1067-3, pp: 738-748.

Harrold, M.J. and M.L. Soffa, 1989. Interprocedual data flow testing. Proceedings of the ACM conference on Software Engineering Notes, December 8, 1989, ACM, New York, USA., ISBN:0-89791-342-6, pp: 158-167.

Harrold, M.J. and M.L. Souffa, 1988. An incremental approach to unit testing during maintenance. Proceedings of the IEEE Conference on Software Maintenance, October 24-27, 1988, IEEE, Scottsdale, Arizona, ISBN: 0-8186-0879-X, pp: 362-367.

Ii, M.F., D. Jin, G. Rothermel and M. Burnett, 2002. Test reuse in the spreadsheet paradigm. Proceedings of the 13th International IEEE Symposium on Software Reliability Engineering, November 3, 2002, IEEE, New York, USA., ISBN: 0-7695-1763-3, pp: 257-268.

Kumar, A. and R. Goel, 2012. Event driven test case selection for regression testing web applications. Proceeding of the International IEEE. Conference on Advances in Engineering, Science and Management, March 30-31, 2012, IEEE, Nagapattinam, Tamil Nadu, ISBN: 978-1-4673-0213-5, pp: 121-127.

Lee, J.A.N. and X. He, 1990. A methodology for test selection. J. Syst. Software, 13: 177-185.

Leung, H.K.N. and L. White, 1990. Insights into testing and regression testing global variables. J. Software Maintenance: Res. Pract., 2: 209-222.

Li, B., D. Qiu, H. Leung and D. Wang, 2012. Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. J. Syst. Software, 85: 1300-1324.

Lin, F., M. Ruth and S. Tu, 2006. Applying safe regression test selection techniques to java web services. Proceeding of the International IEEE. Conference on Next Generation Web Services Practices, September 25-28, 2006, IEEE, Seoul, South Korea, ISBN: 0-7695-2664-0, pp: 133-142.

Martins, E. and V.G. Vieira, 2005. Regression Test Selection for Testable Classes. In: Dependable Computing. Cin, M.D., M. Kaaniche and A. Pataricza (Eds.). Springer Berlin Heidelberg, Berlin, Germany, pp: 453.

Mirarab, S., S. Akhlaghi and L. Tahvildari, 2012. Size-constrained regression test case selection using multicriteria optimization. Software Eng. IEEE. Trans., 38: 936-956.

Orso, A., N. Shi and M.J. Harrold, 2004. Scaling regression testing to large software systems. Proceedings of the ACM Conference on Software Engineering Notes, November 6, 2004, ACM, New York, USA., pp: 241-251.

Rothermel, G. and M.J. Harrold, 1993. A safe, efficient algorithm for regression test selection. Proceedings of the International Conference on Software Maintenance, September 27-30, 1993, Montreal, Canada, pp: 358-367.

Tao, C., B. Li, X. Sun and C. Zhang, 2010. An approach to regression test selection based on hierarchical slicing technique. Proceeding of the 34th Annual IEEE Conference Workshops on Computer Software and Applications, July 19-23, 2010, IEEE, Seoul, South Korea, ISBN: 978-1-4244-8089-0, pp: 347-352.

White, L. and B. Robinson, 2004. Industrial real-time regression testing and analysis using firewalls. Proceedings of the 20th International IEEE. Conference on Software Maintenance, September 11-14, 2004, IEEE, New York, USA., ISBN: 0-7695-2213-0, pp: 18-27.

White, L., H. Almezen and S. Sastry, 2003. Firewall regression testing of GUI sequences and their interactions. Proceedings of the International Conference on Software Maintenance, September 22-26, 2003, Washington, DC., USA., pp: 398-409.

Wong, W.E., J.R. Horgan, S. London and H. Agrawal, 1997. A study of effective regression testing in practice. Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering, November 2-5, 1997, Albuquerque, NM., pp: 264-274.

Zheng, J., L. Williams and B. Robinson, 2007. Pallino: Atomation to support regression test selection for COTS-based applications. Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, November 5-9, 2007, ACM, New York, USA., ISBN: 978-1-59593-882-4, pp: 224-233.