2015 IEEE 39th Annual International Computers, Software & Applications Conference

Modification Impact Analysis based Test Case Prioritization for Regression Testing of Service-Oriented Workflow Applications

Hongda Wang
PLA University of Science and Technology
Nanjing, China
wanghogngda000@126.com

Jianchun Xing*
PLA University of Science and Technology
Nanjing, China
xjc@893.com.cn

Qiliang Yang
PLA University of Science and Technology
Nanjing, China
yql@893.com.cn

Deshuai Han
PLA University of Science and Technology
Nanjing, China
handeshuai@126.com

Xuewei Zhang
PLA University of Science and Technology
Nanjing, China
zxw19880707@163.com

***Abstract*—Test case prioritization for regression testing is an approach that schedules test cases to improve the efficiency of service-oriented workflow application testing. Most of existing prioritization approaches range test cases according to various metrics (e.g., statement coverage, path coverage) in different application context. Service-oriented workflow applications orchestrate web services to provide value-added service and typically are long-running and time-consuming processes. Therefore, these applications need more precise prioritization to execute earlier those test cases that may detect failures. Surprisingly, most of current regression test case prioritization researches neglect to use internal structure information of software, which is a significant factor influencing the prioritization of test cases. Considering the internal structure information and fault propagation behavior of modifications respect to modified version for service-oriented workflow applications, we present in this paper a new regression test case prioritization approach. Our prioritization approach schedules test cases based on dependence analysis of internal activities in service-oriented workflow applications. Experimental results show that test case prioritization using our approach is more effective than conventional coverage-based techniques.**

***Keywords*—test case prioritization; dependence analysis; service-oriented workflow applications; modification impact**

I. Introduction

With the emergence of cloud computing, service-oriented workflow, a large-scale programming mode, has gradually become a mainstream technology for developing instant applications in an open environment [1, 2]. Industrial service-oriented workflow applications may orchestrate web services or other workflow applications to provide value-added service using orchestration language like Web Service Business Process Execution Language (WS-BPEL or BPEL for short). However, these applications usually present some faults or defects, particularly during the evolution of service composition. The maintenance of these applications is expensive. On an average, these activities are account for as much as two-thirds of the overall software life cycle costs [3, 4]. However, the cost of maintenance can be reduced if old test cases and results can be reused to ensure that no previously function has failed as a result of the modification [5, 6]. Therefore, regression testing, one of the most important approaches to provide confidence during software evolution, has gained growing concerns in recent years [7, 8].

Although many researchers have pointed out that frequent execution of regression test is crucial in successful application development [6, 9], rerunning the regression test cases for service-oriented workflow applications may be a long-running and time-consuming (days and even weeks) process. In addition to this, the cost of using a service (for services with access quotas or per-use basis [10]) or the disruptions of using a service (such as stock-exchange systems [11]) also requires to detect failures as soon as possible when executing the test suite. Therefore, we have to resort to the technique of test case prioritization to improve cost-effectiveness of regression testing [12]. Test case prioritization provides a way to schedule tests cases to meet certain objectives. The main objective of test case prioritization is to increase the fault detection rate as early as possible so that can reduce the time and cost in maintaining service-oriented workflow applications. However, it is difficult to know fault detection information until testing is finished. Therefore, most of test case prioritization techniques schedule test cases rely on surrogates and expect that early satisfying these surrogates can lead to increasing the fault detection capabilities [13]. A commonly used surrogate of test case prioritization is the coverage of certain program entities (statements or branches) [6, 14, 15]. However, entity coverage is not sufficient to guarantee a high fault detection rate in some cases [16], which urges us to find a more precise approach to schedule test cases for testing service-oriented workflow applications.

As is pointed out in [17] and [18], the faults of software are usually caused by the propagation of defects with the internal structure of software and thus the design of software internal structure has an important influence on software quality. Therefore, study on the test case prioritization of service-oriented workflow applications should take not only coverage information but also the internal structure information into consideration. Surprisingly, to the best of our knowledge, the internal structure information of service-oriented workflow applications has been inadequately considered in existing regression test prioritization research. Therefore, in this paper, considering both the fault propagation behavior of modifications and internal software structure in service-oriented workflow applications, we propose a new approach to schedule test cases for regression testing.

The internal structure of service-oriented workflow applications can be seen as the interaction of *activities* to realizing the expected target. The interaction of activities include performing process logic, exchanging message, invoking external web services, to name a few. These interactions can be analyzed through conducting dependence analysis between activities in service-oriented workflow applications. The modification or fault in one activity will definitely propagate to other activities that directly or indirectly dependent on it. In this paper, we leverage the *modification impact of activity* to measure the *testing importance of activity* in service-oriented workflow application. Furthermore, combined with the modification information (it can be computed by comparing the difference between original version and its variant), we can derive the *testing importance of test case* by its coverage information. In the same time, according to the testing importance of test case, we schedule test cases in a specific order. To this end, all kinds of BPEL dependencies between activities should be analyzed firstly. Apart from *control dependence*, *data dependence*, and *asyn-invocation dependence* [19], we identify another two program dependences: *correlation dependence* and *synchronization dependence*. With these dependences in mind, we establish a graph called *BPEL activity dependence graph* to quantitatively compute the modification impact of each activity. We conducted experiments using our approach and conventional coverage-based techniques with 8 service-oriented workflow applications. Experimental results demonstrate that the test cases scheduled by our approach can achieve higher fault detection rate.

The rest of this paper is organized as follows. Section Ⅱ introduces some preliminaries. Section Ⅲ shows a running example to motivate our approach. Section Ⅳ presents our approach. Section Ⅴ reports the experiments using our proposal. Section Ⅵ presents some discussions about our approach. Section Ⅶ reviews some related works. Section Ⅷ gives a summary and future work.

II. PRELIMINARIES

This paper addresses the problem of test case prioritization for regression testing in service-oriented workflow applications. Specifically, we take BPEL workflow application, one of the most popular service-oriented workflow applications, as an example to illustrate our approach. For the readers' convenience, in this section, we introduce the problem of test case prioritization and the basics of WS-BPEL language.

A. Test Case Prioritization

Test case prioritization [12 ,20] is one of the most important kind of regression testing technique [13]. With the information gained in previous software evaluation, we can design techniques to rerun the test cases to achieve a certain goal in the regression testing. For example, proper test case prioritization techniques increase the fault detection rate of a test suite and the chance of executing test cases with higher rates of fault detection earlier [20]. We adopt the test case permutation problem from [12] as follows:

Definition 1. (Regression test prioritization problem) *Given a test suite T, a set O containing all permutations of T, and a function f from O to the real numbers, find an o ∈ O such that (∀ o′ ∈ O)[f(o) ≥ f(o')].*

In this paper, the specific purpose is to find an order o of test suite T aiming to increase the fault detection rates. To qualify the fault detection rate of a given test suite, f is always the metric of *Average Percentage of Faults Detected* (*APFD*) function [12]. *APFD* values range from 0 to 1 and a higher *APFD* value indicates a higher fault detection rate. Let T be a test suite containing n test cases, F be a set of m faults revealed by T, and TF_i be the first test case index in ordering o of O that reveals fault i. The following equation gives the *APFD* value for ordering o [12].

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{nm} + \frac{1}{2n} \qquad (1)$$

B. Basics of WS-BPEL Language

WS-BPEL is a language that composes partner Web Services into BPEL workflow applications [6, 21]. It defines the process logic through activities that can be divided into two classes: basic and structured. The former describe the elemental steps of the process behavior; these activities include <receive>, <invoke>, <reply>, and <assign>. The latter comprise the basic activities into structures that express control-flow logic of a BPEL application; these activities include <sequence>, <flow>, <pick>, <switch>, <while>, and <if>. Structured activities can contain recursively other basic and/or structured activities. WS-BPEL provides two communication mechanisms to invoke web services: asynchronous (one-way <invoke>) and synchronous (request-response <invoke>). WS-BPEL describes concurrency and synchronization mechanism using structured activity <flow>. The synchronization mechanism is expressed using <link>.

WS-BPEL language provides correlation mechanism to guarantee that a SOAP message will reach a specific instance with the right state and history for interaction [29]. This mechanism allows an infrastructure that conforms to WS-BPEL to use correlation tokens to provide instance routing automatically, provided that the message structure is well defined. The use of correlation tokens is restricted to message parts described in this manner.

III. RUNNING EXAMPLE

In this section, we use a BPEL workflow application: *travel agency*, which is adapted from *Travel*, as a running example to motivate our approach. This is a so well-known and carefully designed application that it has been used by many researches [19, 22]. First, we give a briefly overview about this application.

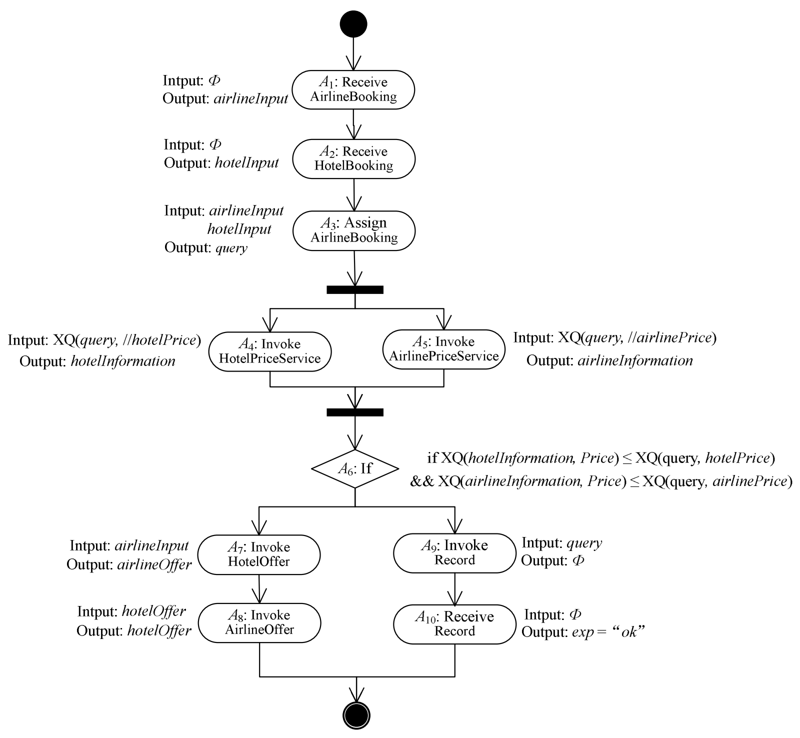

Intput: Φ
Output: airlineInput
A1: Receive AirlineBooking
Intput: Φ
Output: hotelInput
A2: Receive HotelBooking
Intput: airlineInput hotelInput
Output: query
A3: Assign AirlineBooking
Intput: XQ(query, //hotelPrice)
Output: hotelInformation
A4: Invoke HotelPriceService
A5: Invoke AirlinePriceService
Intput: XQ(query, //airlinePrice)
Output: airlineInformation
A6: If
if XQ(hotelInformation, Price) ≤ XQ(query, hotelPrice) && XQ(airlineInformation, Price) ≤ XQ(query, airlinePrice)
Intput: airlineInput
Output: airlineOffer
A7: Invoke HotelOffer
A9: Invoke Record
Intput: query
Output: Φ
Intput: hotelOffer
Output: hotelOffer
A8: Invoke AirlineOffer
A10: Receive Record
Intput: Φ
Output: exp = "ok"

Fig. 1. A running example: the *travel agency* application.

For intuitive expression, we use UML activity diagrams instituting of BPEL codes (in XML format) to depict these applications. Fig. 1 shows the BPEL workflow application of *travel agency*. In each activity diagram, a node represents a BPEL activity and an edge represents a transition between two activities. In addition, we also annotate the nodes with extracted application information, such as input and output parameters of activities or any XPath Query [14 , 23] used by the activities in this BPEL workflow application. In addition, we number the nodes as A_1, A_2,⋯, A_{10} to ease subsequent discussions. The following is a detailed explanation for the application *travel agency*. On receiving ordering information from client by activities A_1 and A_2, the application of *travel agency* concurrently invoke the *HotelPriceService* and *AirlinePriceService* to inquire the price of hotel and airline (activities A_4 and A_5). If the price of hotel and airline is equal or less than inputted price by client (verified by activity A_6), this application will execute the activities A_7 and A_8 to book a hotel and an airline ticket. Otherwise, this workflow application will invoke *RecordService* to record the failed booking information about client.

It is noted that this version of BPEL workflow application is modified from the initial version. In this version, assuming that two activities (A_7 and A_9) have been revised compared with the last version. Let us apply the commonly-used approaches (statement coverage-based or branch coverage-based) [6, 20] to this scenario. We find that the test cases corresponding to activities A_7 and A_9 in this application are arranged randomly as the two activities provide the same coverage information. However, in our approach, we find that the modification impact of activities A_7 and A_9 are different and activity A_9 is larger. Therefore, the test cases corresponding to activity A_9 should be rerun in advance than test cases corresponding to activity A_7. In this paper, we will propose a more precise approach for arranging the test cases of BPEL workflow applications in regression testing.

IV. Our Test Case Prioritization

In this section, we present a more precise test case prioritization approach for service-oriented workflow applications, which is based on the analysis of dependence relation among activities. Given a test suite T for a service-oriented workflow application, our target is to reorder T according to the sum of testing importance of modified activities covered by each test case. Before the specific description of our approach, we firstly define the importance of activities and test cases.

A. Testing Importance of Activity

To meet the ever-expanding needs of users, service-oriented workflow applications must be evolving and some modifications or changes are inevitable. Therefore, it is vital for organizations to perform software maintenance (such as regression testing) in such a way as to reduce the potential new bugs to be introduced by the modifications. Most of exiting techniques addressing the problem of test case prioritization are based on the coverage information [6, 20]. However, as is reported in [24 , 25], with the increasing complexity of software systems, the internal structure of software is becoming one of the most important factors that influencing quality of the final software products. However, none of these techniques [6, 20] considered this factor. Modification impact analysis in internal structure of software is important because it can also assist in determining the consequences of the modification(s) [26]. Program traces and module dependency techniques are performed to determine which modules must be changed along with the target module [26, 27]. In this paper, we leverage module dependency technique to quantitatively evaluate the change impact of modified activities in modified version of BPEL workflow applications. Our approach is based on the following two assumptions:

1) the modification or fault in one activity will definitely propagate to other activities that are dependent on the changed or faulty activity directly or indirectly in *BPEL activity dependence graph* we proposed;

2) all activities have the same probability 100% that they may be affected by the modified or faulty activity.

We analyze the modification impact of activities with the dependence relation between each two activities. To this end, all kinds of program dependences between activities in BPEL workflow applications should be analyzed first. Apart from *control dependence*, *data dependence* [28] and *aysn-invocation dependence* [19], we identify another two program dependences in BPEL workflow application: *correlation dependence* and *synchronization dependence*. We capture all these 5 kinds of dependences in the BPEL activity dependence graph we defined. Based on these intermediate representation, both direct and transitive dependence between any two activities can be identified conveniently. To make this paper self-contained, we will firstly introduce the concept of control dependence, data dependence and aysn-invocation dependence.

Informally, an activity A_j is control dependent on activity A_i if and only if A_i represents the predicate of a conditional branch or entry activity (assume that every BPEL workflow application has an entry activity) that directly controls whether A_j is executed. Taking the BPEL workflow application in Fig. 1 as an example. All of the activities of $A_1 \sim A_6$ in the BPEL workflow application of Fig. 1 are control dependent on entry activity. Data dependence can be classified into three categories: *true dependence* (*def-use dependence*), *anti-dependence* (*use-def* dependence) and *output dependence* (*use-use* dependence) [17]. An activity A_j is def-use dependent on activity A_i if and only if a variable defined at A_i and is used at A_j. An activity A_j is use-def dependent on activity A_i if and only if a variable used at A_i and is defined at A_j. An activity A_j is use-use dependent on activity A_i if and only if a variable used at A_i and is also used at A_j. In the BPEL workflow application of Fig. 1, activity A_3 is true dependent on activity A_2 because A_2 uses a variable *hotelInput* which is defined by A_2. Anti-dependence and output dependence are not appear in this workflow application. In fact, anti-dependence and output dependence can be avoided through variable renaming. Therefore, data dependence defined in this paper only refers to true dependence. Asyn-invocation dependence is caused by the asynchronous communication mechanism [19]. An activity A_j is asyn-invocation dependent on activity A_i if and only if A_j is a <receive> activity which is responsible for receiving the response of a preceding one-way <invoke> activity A_i. Taking the BPEL workflow application in Fig. 1 as an example, Activity A_{10} is asyn-dependent on activity A_9 because A_{10} is responsible for receiving the response of A_9. These three dependences are captured in this paper.

Apart from control dependence, data dependence and asyn-invocation dependence, we identify another two BPEL program dependences. One of the two dependences is caused by the correlation mechanism among *reception* activities in BPEL workflow application. The reception activities include <receive>, request-response <invoke> and <onMessage>. According to WS-BPEL 2.0 [21 , 29], two properties (i.e., the property *createInstance* and property *correlation*) of a reception activity in a BPEL workflow application play central roles in message routing. A start activity of a BPEL workflow application is one reception activity whose value of *createInstance* property is "*yes*". Let us consider a scenario: a <receive> activity is a start activity of a BPEL workflow application, then an instance of the application is created and all the other messages received by reception activities corresponding to this instance are routed to this newly created instance. We find that there are no control, data and asyn-invocation dependencies between the <receive> activity and one of other reception activities, which has the same correlation set with the <receive> activity, in this BPEL workflow application. However, the two activities do have a certain kind of dependence since it could lead to a deadlock if the execution order of the two activities is reversed. The informal definition of this dependence is given as follows.

Definition 2. (Correlation dependence) *An activity A_j is correlation dependent on activity A_i if and only if both A_i and A_j are reception activities in a BPEL workflow application share the common property and A_i is the start activity whose value of createInstance property is "yes".*

As is shown in Fig. 1, activities A_1, A_2, A_4, A_5, $A_7 \sim A_{10}$ are reception activities in this BPEL workflow application. Activity A_1 is a start activity and its property of *createInstance* is set to "*yes*". Furthermore, each message in the operation of activities A_1, A_2, A_4, A_5, $A_7 \sim A_{10}$ is correlated by correlation set *ID* (identity card number) to ensure these reception activities interact with same client. Therefore, activities A_2, A_4, A_5, $A_7 \sim A_{10}$ are correlation dependent on activity A_1.

In addition, The activity <flow> is a structured activity that describes concurrency and synchronization mechanism. A set of activities included in a <flow> activity can be executed in any order (except for the happen-before relationship defined by the <sequence>). A <flow> activity is complete if and only if all activities included in it have been completed. WS-BPEL language expresses synchronization dependences between activities by <link>s. The two activities connected by <link> do have a certain kind of dependence because the execution order of the two activities is cannot reversed. Informally, the definition of this dependence is given as follows.

Definition 3. (Synchronization dependence) *An activity A_j is synchronization dependent on activity A_i if and only if both A_i and A_j are included in <flow> activity and activity A_j is the target of a <link> that has activity A_i as the source.*

In Fig. 1, activities A_4 and A_5 are included in <flow> activity. The two activities can be executed in any order and are not dependent on each other.

In this paper, we capture these 5 kinds of dependences in a BPEL activity dependence graph where nodes denote basic or structured activities except for Entry node and the edge indicates a certain dependence between two nodes. In order to accurately describe BPEL workflow application, each control dependence edge is labeled with True (T) or False (F). Additionally, the data dependence edges should be attached to correlated variables, but the labels are omitted in order to conveniently denote them. Formally, we give the following definition of BPEL activity dependence graph.

Definition 4. (BPEL Activity Dependence Graph) *A BPEL activity dependence graph is a directed graph <N, E>, where*

- *N is a set of nodes. There is one entry node in N while the other nodes denote the basic or structured activities.*

- *$E \subseteq N \times N$ is a set of directed edges, and an edge <X, Y> $\in$ E directed from X to Y denotes the control dependence, data dependence, asyn-invocation dependence, correlation dependence or synchronization dependence between any two activities represented by X and Y.*

Fig. 2 shows the BPEL activity dependence graph for the BPEL workflow application of *travel agency* in Fig. 1. As is depicted in Fig. 2, different colored arrows denotes different dependences. It is noted that there is no synchronization dependence in this BPEL workflow application.

In this paper, we leverage module dependency technique to quantitative evaluate the modification impact of activities in modified version of BPEL workflow applications. The modification or fault in one activity will

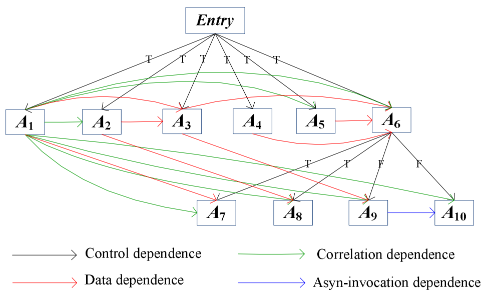


Fig. 2. BPEL activity dependence graph for the travel agency application.

definitely propagate to other activities that directly or indirectly dependent on it in BPEL activity dependence graph. Based on the assumption mentioned above, we use program slicing, one of the most useful techniques for simplifying programs by focusing on selected aspects of computing, to compute the impact of modified activities. The process of program slicing deletes the parts of the program which have no effect upon the aspects of interest [30]. For an activity A in a BPEL activity dependence graph P, the slice with respect to the slicing criterion $< P, A>$ includes only those activities in P needed to capture the computing of A at P. In other words, the slice of BPEL activity dependence graph includes those activities that are directly or indirectly dependent on the activity A. In this paper, we only construct the forward slice (the other one is backward slice). A forward slice contains those activities of the BPEL activity dependence graph which are affected by the slicing criterion [31]. Modification impact of an activity influences the activities that are directly or indirectly dependent on it. Informally, we give the following definition of *Modification Impact of Activity*.

Definition 5. (Modification Impact of Activity, MIA) *Given an activity M in a BPEL workflow application, the modification impact of activity M is a forward slice of activity M in BPEL program dependence graph G, that is:*

$$MIA(M) = ForwardSlice(G, M) \tag{2}$$

Based on the modification impact, we give the concept of *Testing Importance of Activity* (*TIA*). *TIA* is the sum of activities influenced by the modified activity in BPEL workflow application. Informally:

Definition 6. (Testing Importance of Activity, TIA) *Given an activity M in a BPEL workflow application, the testing importance of M [TIA(M)] is the total number of activities in IMA(M) in BPEL program dependence graph G, that is:*

$$TIA(M) = \sum_{M} MIA(M) \tag{3}$$

Let us take the BPEL activity dependence graph in Fig. 2 as an example to illustrate our definitions. If we make a forward slice with the activity A_6, the $MIA(A_6)$ is shown in Fig. 3. Therefore, the $TIA(A_6) = 5$.

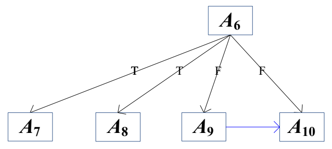


Fig. 3. The $TIA(A_6)$ in BPEL activity dependence graph for travel agency

B. *Our prioritization approach*

Based on the *Testing importance of Activities* (*TIA*) defined in the last sub-section, combined with the modified information, we define the importance of test cases, i.e., *Testing importance of Test Case* (*TITC*). Informally:

Definition 7. **(Testing Importance of Test Case, TITC)** *Given a test case t ∈ T for a BPEL workflow application, the testing importance of t [TITC(t)] is defined by the testing importance of modified activities covered by test case t. Let the function gtc(t) returns the set of modified activities covered by t, TITC(t) is the total number of TIC(M) {M ∈gtc(t)}, that is:*

$$TITC(t) = \sum_{M \in gtc(t)} TIC(M) \tag{4}$$

Based on the *TITC* we defined above, we propose our approach of test case prioritization for BPEL workflow applications. The framework of our approach can be summarized in Fig. 4, specifically:

(1) Given a modified BPEL workflow application V_2, all kinds of BPEL activity dependencies we defined in the last subsection between activities should be analyzed, and therefore the BPEL activity dependence graph (BADG) corresponding to this BPEL workflow application is established.

(2) By analyzing the differences between BPEL workflow application V_1 and its modified version V_2, we can locate the modified activities and derive the modified information (*modifiedSet*(V_2-V_1)).

(3) Computing the *Testing importance of activities* (*TIC*) in *modifiedSet*(V_2-V_1) in the BADG, which is established in step (1).

(4) Computing the *Testing importance of test case* (*TITC*) of all the test cases according to the modified activities covered by it. Furthermore, we prioritize the order of test cases according to its corresponding *TITC* value from largest to smallest. It is noted that if the *TITC* value of two activities is equivalent, we randomly arrange the two activities. Therefore, we can derive a specific order of regression test cases.

Let us return to the motivating example in Fig. 1. Activities A_7 and A_9 have been revised compared with the last version. Therefore, the *modifiedSet*(V_2-V_1) = { A_7, A_9}. Furthermore, *TIA*(A_7) = 1 and *TIA*(A_9) = 2. In this application, let us assume that the corresponding regression test cases to test A_7 and A_9 are t_1 and t_2 respectively. Therefore, *TITC*(t_1) = 1 and *TITC*(t_2) = 2. Thus, t_2 should be rerun in advance.

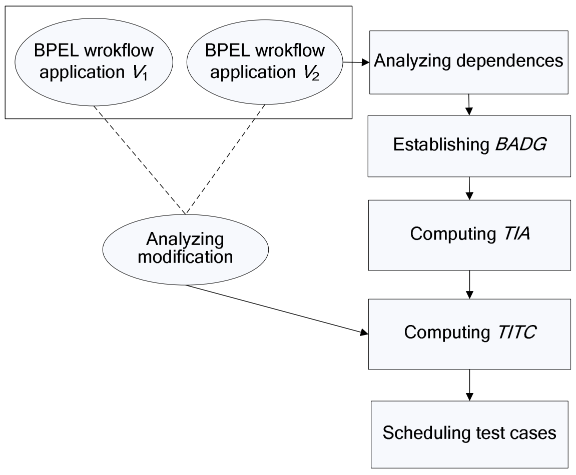


Fig. 4. The framework of our approach.

V. EXPERIMENTS

We conduct several experiments in this section. Furthermore, we compare our approach with several typical approaches in terms of *APFD*.

A. Experimental setup

We use 8 BPEL workflow applications to evaluate the effectiveness of our approach. These 8 applications come from some popular BPEL engines (such as ActiveBPEL, IBM BPWS4J, etc.) and BPEL specification (Tab. Ⅰ). These applications are also used in other BPEL test case generation [27, 28] or regression test prioritization [29, 30]. We use A to H to represent the corresponding BPEL workflow applications. The columns “Application,” “Source,” “Element,” and “LOC” represent the name of BPEL applications, the source of BPEL applications, the number of XML elements, and the number of lines of code of each application, respectively. In addition, the columns “COR” and “Link” depict whether or not correlation sets and links have been used in the BPEL workflow applications. In our experiment, all BPEL workflow applications and their partner web services run in the ActiveBPEL (Version 4.1) engine [31]. They form the basis of our experiments.

TABLE I. RELEVANT INFORMATION OF THE EXPERIMENTAL APPLICATIONS

Ref.	Application	Source	Element	LOC	COR	Link	Versions
A	Travel	Oracle BPEL Process Manager	39	90	No	No	9
B	ATM	ActiveBPEL	94	180	Yes	No	7
C	GYMLocker	BPWS4J	23	52	Yes	No	6
D	LoanApproval	ActiveBPEL	41	102	No	Yes	6
E	MarketPlace	BPWS4J	31	68	Yes	No	9
F	Auction	BPEL Specification	50	138	Yes	No	7
G	RiskAssessment	BPEL Specification	44	118	No	Yes	8
H	Loan	Oracle BPEL Process Manager	55	147	No	No	7

For evaluation, we need some modified versions of these BPEL workflow applications. However, a few modified versions have been published by the developers. Therefore, to continue our experiments, we invite some research partners (non-authors) to make some modifications to the 8 original applications. To guarantee the independence of modification, each version only involves one modification or fault. Following the spirit of mutation testing [32] to seed faults in artifacts, we partition faults into three categories, namely, BPEL, WSDL, and XPath. The faults which are seeded into BPEL are mainly some logic faults or some spelling faults related to execution result. The faults which are seeded into WSDL are mainly incorrect types, messages, port types, and bindings (e.g., incorrect XML Schema definition, incorrect operation definition or misusage of input/output message). An XPath fault is the wrong usage of XPath expressions, such as extracting the wrong content, or failing to extract any content. A similar method was also used by Mei et al. [6] and Ni et al. [29]. As Table depicts that we created 59 versions totally. Strictly following the methodology in [14], we generated test cases to construct test pools. For each base BPEL application, 100 test cases were generated. For each application, we ran all test cases in the test suite and saved the output and test traces. We then added test cases to the test pool to ensure that each activity in every base BPEL application was exercised by at least 5 test cases. Finally, we obtained 30 test cases for each base BPEL workflow application.

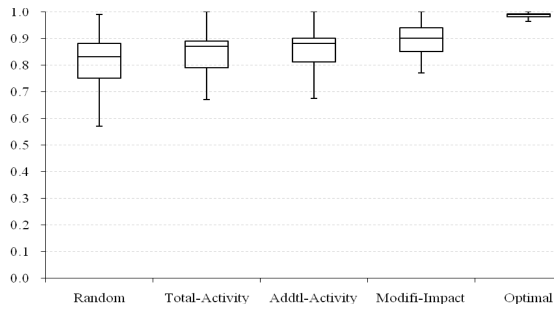
To illustrate the advantages of our approach, we conducted experiments for the following techniques:

- Random ordering (denoted as **Random**): This technique schedules the test cases randomly [12].
- Total Activity Coverage Prioritization (denoted as **Total-activity**): This technique is adapted from [12], which sorts the test cases in the descending order of the total number of activities that each test case has covered.
- Additional Activity Coverage Prioritization, denoted as **Addtl-activity**): This technique is also adapted from [12], which sorts the test cases iteratively selects a test case that yields the greatest cumulative activity coverage.
- Our approach (i.e., Modification impact analysis based approach, denoted as **Modification-impact**).
- Optimal ordering (denoted as **Optimal**): This technique iteratively selects test case by the ability of exposing the most faults not yet exposed [12]. However, as is pointed by [12] and [33], this technique is unrealistic because it needs to know which test cases will reveal the particular fault. In our experiments, it can be served as a reference of an upper bound of a test case prioritization.

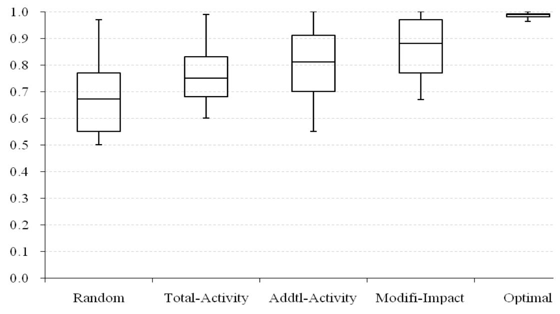
B. Results and analysis

To analyze the effectiveness of our approach, we apply the 5 methods (include ours) mentioned above to the modified versions of the 8 BPEL workflow applications respectively and the calculate *APFD* values (by comparing the value TF_i of each test suite on each fault version) for each base BPEL workflow application. We repeated each experiment 10 times using the generated test suites and average the results over these runs. Fig. 5 shows the results using box-whisker plots. For each box-whisker plot, the horizontal axis represents the test case prioritization techniques mentioned above and the vertical axis represents *APFD* values. A to H represent the corresponding individual BPEL workflow applications in Tab. Ⅰ.

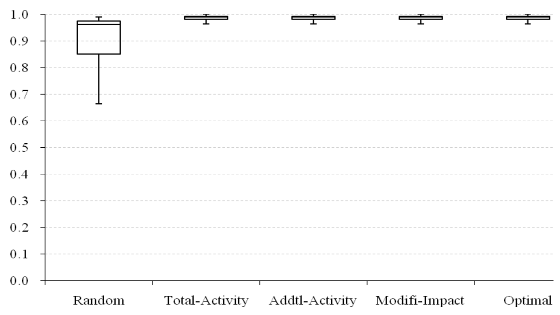
As is shown in Fig. 5, the Random, Total-activity, Addtl-activity, Modifi-impact and Optimal are the methods mentioned in the last subsection. The graph shows that for each of the 8 base BPEL workflow applications, the *APFD* of test cases prioritized by these 5 approaches, on average, is over the set of modified versions. The graph denotes that, for this experiment, the 3 approaches (Total-activity, Addtl-activity, and Modification-Impact) improve the values of *APFD* compared with Random. In fact, for some cases (base BPEL workflow applications C and F), the values of *APFD* detected by the three approaches (Total-activity, Addtl-activity, and Modifi-Impact) are nearly same for the modified versions of the base BPEL workflow applications. In addition, we observe that the values of *APFD* detected by Modifi-Impact are higher than Total-activity and Addtl-activity for most cases.



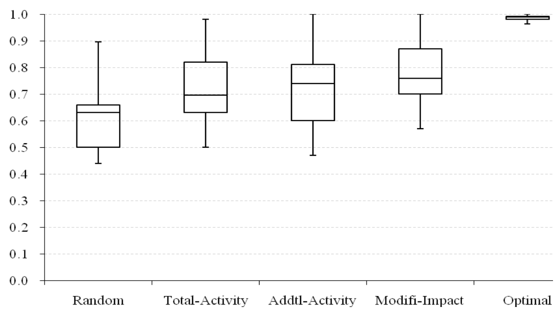
(A) Travel



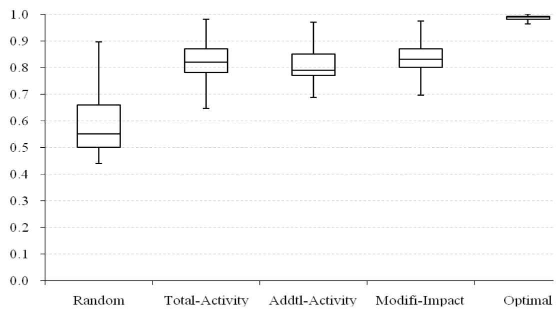
(B) ATM



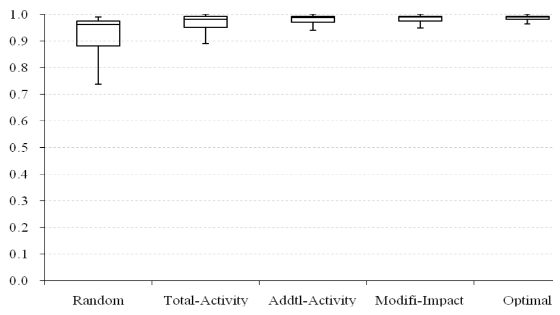
(C) GYMLocker



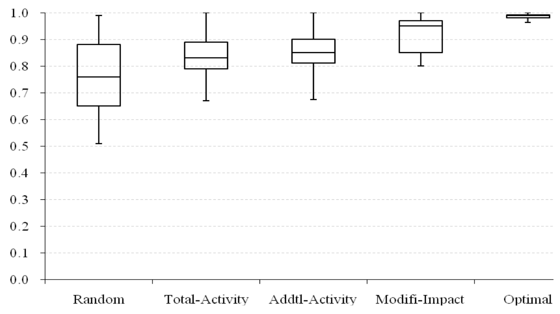
(D) LoanApproval



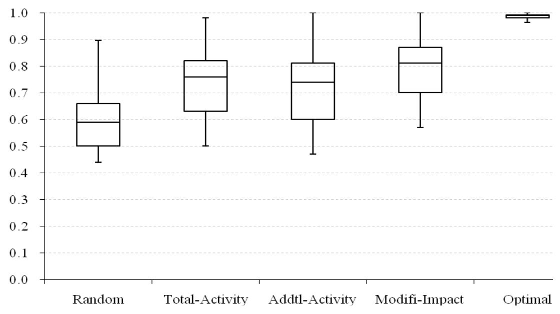
(E) MarketPlace



(F) Auction



(G) RiskAssessment



(G) Loan

Fig. 5. Comparisons on each BPEL workflow application using *APFD* measurement.

C. *Threats to Validity*

In our experiments, the external validity mainly originates from the following aspects. First, this paper only chooses some typical BPEL workflow applications to test the feasibility of our approach. The test results show that our approach is promising. However, we only use a limited number of BPEL workflow applications and certain types of modified versions in our experiments. Like most other empirical studies, the result of our empirical study may not be generalized to cover all cases. With this consideration in mind, we plan to apply our approach to large-scale, evolving BPEL applications in the future. Second, we conduct our experiments in the ActiveBPEL engine (version 4.1). However, some features and communication model are different with respect to BPEL engines. To reduce these threats, we plan to conduct experiments on different platforms in the future.

VI. Discussion

In this study, we have proposed an approach that schedules test cases for service-oriented workflow applications. Although the test cases ordered by our approach outperforms traditional approaches and can test the modified version of BPEL applications effectively, some shortcomings or weaknesses still exist. In this subsection, we give a discussion in detail.

One of the major disadvantages of our approach is the increase in the workload of BPEL activity dependence graph analysis and slicing with the scale of BPEL applications. In this study, we use some available and small BPEL applications we can find to experiment our approach and the result demonstrate that our approach is promising. However, with the scale of BPEL applications, the workload will increase in some certain. In the future studies, we would try to find a more efficient algorithm to trade off the effectiveness and efficiency on these.

In this paper, to quantitative evaluate the change impact of modified activities, we propose two assumptions. We assume that all activities have the same probability 100% that they can be affected, and that a modification in one activity will definitely propagate to other activities that are dependent on it in BPEL activity dependence graph. This may not meet the practice. This is one future work we will address.

Another disadvantage of our approach is that it only covers some basic structures and elements of WS-BPEL 2.0. Some advanced activities, e.g., <scope> with fault handing, which are designed to undo the partial and unsuccessful work of a <scope> in which a fault has occurred, have not been fully considered so far. This is one future work we need to consider.

Otherwise, the approach proposed in this paper for scheduling test cases from original test suites respect to original version for modified service-oriented workflow applications is applied to BPEL workflow applications. However, our approach is not limited to BPEL workflow applications. In practice, the approach described in this paper can also be applied to other service-oriented workflow applications (such as OWL-S applications, executable WS-CDL applications, flow-driven Mashup applica-tions, etc.).

VII. Related work

Regression testing has been recognized as an effective technology to ensure the quality of software after a system has been changed. It mainly includes regression test selection, regression test prioritization and regression test minimization, etc. Many interesting approaches or techniques have been done around these problems [6, 10, 11, 14, 15, 33, 34]. In this section, we review some related work about regression test case prioritization for BPEL workflow applications and compare them with our work. For more information about testing and verification in service-oriented architecture, please refer to [11].

In this paper, we classify the approaches that address the problem of test case prioritization for BPEL workflow applications into two categories. The first category assumes that the testing environment context is static. In this way, the partner web service invoked by the BPEL workflow application is no change for the regression testing. For example, Mei et. al [6, 15] proposes a set of roles to support service-oriented regression testing and outlines a scenario to illustrate how to collaborate these roles. They mainly studies whether the WSDL and XPath information may facilitate effective regression testing of service-oriented workflow applications. Hou et. al [10] propose a technique to address the problem of test case prioritization for considering quotas to requests for specific external web services. In their testing environment, the requests for partner web service constrain a certain total amount. Zhai et al. [35] use the dynamic features of service selection to reduce the service invocation cost, and they further study [36, 37] different diversity strategies to reorder test cases. However, these approaches do not consider the internal structures of service-oriented workflow applications. Our approach proposed in this paper falls into this category. In this paper, we propose a test case prioritization approach considering both the internal structure and modification impact of activities. In fact, these approach and our approach complement each other. It would be interesting to combine our approach with these approaches to address the problem of test case prioritization for service-oriented workflow applications.

The second category assumes that the testing environment context is volatile. In this way, the partner web service invoked by the BPEL workflow application can be different for the regression testing. For example, Mei et. al [15] propose a novel strategy that reschedules test cases in a planned regression test session based on the changes of the service under test detected in the course of each actual regression test session. They also propose three particular strategies, which can be integrated with existing test case prioritization approaches. However, their approach also not considers the modification impact of internal activities in service-oriented workflow applications.

Since the test case generation is the basis of regression test prioritization, we review some representative approaches in the area of BPEL workflow applications in recent times. Mei et al. [23] address the integration issues that might be caused by XPath in BPEL workflow applications such as extracting wrong data from an XML message. In addition, Mei et al. propose a set of test coverage criteria to test WS-BPEL applications from the perspective of data flow. They mainly focused on the XPath, which is used to manipulate XML documents. They developed a graph model known as XPath Rewriting Graph (XRG) to capture how an XPath can be rewritten from one form to another in a stepwise fashion. Ni et al. [10, 29] generated a message sequence based on Message Sequence Graph (MSG), which has the information of message sequences and message parameters. This approach mainly considers correlation mechanism and generates message sequence based on random testing.

VIII. Conclussion

Test case prioritization for regression testing is an approach that schedules test cases in a specific order to detect faults early, which is well known as an effective technology to ensure the quality of modified service-oriented workflow applications. In this paper, considering the internal structure and fault propagation behavior of activity in service-oriented workflow applications, we present a new regression test selection approach. Compared with the conventional approaches, our approach can achieve higher fault detection rates. We show the validity and feasibility of our approach through empirical studies.

Our current work only focuses on regression test prioritization problem of service-oriented workflow applications in static testing environment. In the future, we will broaden our work to solve regression test prioritization problem in volatile testing environment.

Acknowledgment

The authors are very grateful to the anonymous reviewers for their insightful remarks and helpful comments on an earlier draft. This work was partially supported by the National Natural Science Foundation of China under Grant NO. 61321491 and the Foundation of State Key Laboratory for Novel Software Technology under Grant NO. KFKT2014B12.

References

[1] Canfora, G., & Di Penta, M. (2009). Service-oriented architectures testing: A survey. In Software Engineering (pp. 78-105). Springer Berlin Heidelberg.

[2] Fu, X., Bultan, T., & Su, J. (2004, May). Analysis of interacting BPEL web services. In Proceedings of the 13th international conference on World Wide Web (pp. 621-630). ACM.

[3] Beizer B. Software testing techniques[M]. Dreamtech Press, 2003.

[4] Leung, H. K., & White, L. (1989, October). Insights into regression testing [software testing]. In Software Maintenance, 1989., Proceedings., Conference on (pp. 60-69). IEEE.

[5] M. E. Ruth and S. Tu. Towards automating regression test selection for Web services. In Proceedings of the 16th International Conference on World Wide Web (WWW 2007), pages 1265–1266. 2007.

[6] L. Mei, Z. Zhang, W. K. Chan, and T. H. Tse. Test case prioritization for regression testing of service-oriented business applications. In Proceedings of the 18th International Conference on World Wide Web (WWW 2009), pages 901–910. ACM, New York, NY, 2009.

[7] Epifani, I., Ghezzi, C., & Tamburrelli, G. (2010, November). Change-point detection for black-box services. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (pp. 227-236). ACM.

[8] Zhu, H., & Zhang, Y. (2012). Collaborative testing of web services. Services Computing, IEEE Transactions on, 5(1), 116-130.

[9] A. K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. Communications of the ACM, 41 (5): 81–86, 1998.

[10] Hou, S. S., Zhang, L., Lan, Q., Mei, H., & Sun, J. S. (2009, August). Generating effective test sequences for BPEL testing. In Quality Software, 2009. QSIC'09. 9th International Conference on (pp. 331-340). IEEE.

[11] Bozkurt, M., Harman, M., & Hassoun, Y. (2013). Testing and verification in service - oriented architecture: a survey. Software Testing, Verification and Reliability, 23(4), 261-313.

[12] Elbaum S, Malishevsky A G, Rothermel G. Test case prioritization: a family of empirical studies [J]. IEEE Transactions on Software Engineering, 2002, 28(2): 159-182.

[13] Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a sur-vey. Software Testing, Verification and Reliability22(2), 67–120 (2012)

[14] L. Mei, W.K. Chan, T.H. Tse, and R.G. Merkel, "XML-manipulating test case prioritization for XML-manipulating services," Journal of Systems and Software, vol. 84, no. 4, pp. 603–619, 2011.

[15] L. Mei, W.K. Chan, and T.H. Tse, B. Jiang, K. Zhai, "Preemptive Regression Testing of Workflow-based Web Ser-vices," IEEE Transactions on Services Computing, DOI : 10.1109/TSC.2014.2322621, 2014.

[16] Marick, B.: The craft of software testing: subsystem testing including object-based and object-oriented testing. Prentice-Hall, Inc. (1994)

[17] Maggie H, Katerina G P. Common trends in software fault and failure data [J]. IEEE Transactions on software engineering. 2009, 35(4):484-496.

[18] Zhou Y M, Leung H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults [J]. IEEE Transaction on Software Engineering, 2006, 32(10): 771-789.

[19] Song, W., Ma, X., Cheung, S. C., Hu, H., Yang, Q., & Lu, J. (2011, July). Refactoring and publishing WS-BPEL processes to obtain more partners. In Web Services (ICWS), 2011 IEEE International Conference on (pp. 129-136). IEEE.

[20] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing tes cases for regression testing. IEEE TSE, 27 (10): 929–948, 2001.

[21] "WS-BPEL 2.0 Specification" 2007. [Online]: Available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[22] Van Der Aalst, W. M., Dumas, M., Gottschalk, F., Ter Hofstede, A. H., La Rosa, M., & Mendling, J. (2008). Correctness-preserving configuration of business process models. In Fundamental Approaches to Software Engineering (pp. 46-61). Springer Berlin Heidelberg.

[23] L. Mei, W. Chan, and T. Tse, "Data Flow Testing of Service-Oriented Workflow Applications," in Proceedings of the 30th International Conference on Software Engineering (ICSE'08), pp. 371-380, New York, NY, USA, 2008.

[24] Abdelmoez W, Shereshevsky M, Gunnalan R, Ammar H H, Yu B, Bogazzi S, Korkmaz M, Mili A. Quantifying software architectures: An analysis of change propagation probabili-ties. In Proc . t he 3rd AC S /IEEE In t e r n at i on al C on f e re n ce on C omput e r S y st e m s an d A ppli cat i on s, Cairo, Egypt, Jan. 3-6, 2005, pp.687-694.

[25] MacCormackA,RusnakJ,BaldWinCY.Exploringthe structure of complex software designs: An empirical study of open source and proprietary code. Management Science, 2006, 52(7): 1015-1030.

[26] N. Nurmuliani, D. Zowghi, S.P. Williams, Using card sorting technique to classify requirements change, in: Proceedings of the 12th IEEE International Requirements Engineering Conference, 2004, pp. 240–248.

[27] S.A. Bohner, Software change impacts-an evolving perspective, in: Proceedings of the International Conference on Software Maintenance, Montreal, Quebec, Canada, 2002, pp. 263–272.

[28] Ferrante, J., Ottenstein, K. J., & Warren, J. D. (1987). The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems (TOPLAS), 9(3), 319-349.

[29] Y. Ni, S.-S. Hou, L. Zhang, J. Zhu, Z.J. Li, Q. Lan, H. Mei, and J.-S. Sun, "Effective message-sequence generation for testing BPEL programs," IEEE Transactions on Services Computing, vol. 6, no. 1, pp. 7–19, 2013.

[30] T. Reps and W. Yang, "The semantics of program slicing and program integration," pp.360-374 in Proceedings of the Colloquium on Current Issues in Programming Languages, (Barcelon% Spain,March 13-17, 1989), Lecture Notes in Computer Science Vol. 352, Springer-Verlag, New York NY(March 1989)

[31] B. W. Xu, J. Qian, X. F. Zhang, Z. Q. Wu, and L. Chen, A Brief Survey of Program Slicing.ACM SIGSOFT Software Engingeering, 2005, 30(2): 1–36 .

[32] Andrews, J. H., Briand, L. C., & Labiche, Y. (2005, May). Is mutation an appropriate tool for testing experiments?[software testing]. In Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on (pp. 402-411). IEEE.

[33] Jiang, B., Zhang, Z., Chan, W. K., & Tse, T. H. (2009). Adaptive random test case prioritization. In Proceedings ASE 2009 (pp. 233–244).

[34] Li, B., Qiu, D., Leung, H., & Wang, D. (2012). Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. Journal of Systems and Software, 85(6), 1300-1324.

[35] K. Zhai, B. Jiang, and W.K. Chan, "Prioritizing test cases for regression testing of location-based services: metrics, tech-niques and case study," IEEE Transactions on Services Computing, 7(1): 54-67, 2014,

[36] K. Zhai, B. Jiang, W.K. Chan, and T.H. Tse, "Taking advantage of service selection: a study on the testing of location-based web services through test case prioritization," Proceedings of the IEEE International Conference on Web Services (ICWS '10), pp. 211–218, IEEE Computer Society, 2010.

[37] L. Mei, Y. Cai, C. Jia, B. Jiang, W.K. Chan, Z. Zhang, and T.H. Tse, "A Subsumption Hierarchy of Test Case Prioritization for Composite Services," IEEE Transactions on Services Computing, DOI: 10.1109/TSC.2014.2331683, 2014.