IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 7, 2022, accepted May 20, 2020, date of publication May 29, 2022, date of current version June 10, 2022.

Bug Tracking for improving software Reliability

HONGDA WANG[1], MAN YANG[1], LIHUA JIANG[1], JIANCHUN XING[2], (Member, IEEE), QILIANG YANG[2], (Member, IEEE), AND FUYONG YAN[3]

[1]Naval Logistics Academy, Tianjin 300450, China
[2]Army Engineering University of PLA, Nanjing 210007, China
[3]PLA 32752 Troops, Dalian 116017, China

Corresponding author: Man Yang (18795851002@163.com)

This work was supported by the Natural Science Foundation of Jiangsu Province under Grant BK20151451.

ABSTRACT Test case prioritization is a method to prioritize test cases to improve the testing efficiency of service-oriented workflow applications. Existing prioritization methods prioritize test cases in different application environments according to different metrics (for example, statement coverage, and path coverage). Web services are orchestrated by service-oriented workflow applications to provide different functions, especially in cloud-based mobile systems. As a result, those applications need more precise scheduling to run test cases that can detect faults earlier. Unfortunately, most regression test case prioritization studies in service-oriented workflow applications neglect the use of activity dependency, which is an important factor that affects test case prioritization. By analyzing the dependences between activities, the modification effects of activities on the modified version of service-oriented workflow applications are calculated. On this basis, this paper proposes a new prioritization method for regression test cases. Experimental results show that our method is more effective than the traditional coverage-based technique in testing case priority.

INDEX TERMS Cloud-based mobile systems, test case prioritization, activity dependency, modification impact.

I. INTRODUCTION

With the advent of service computing and cloud computing, large-scale programming has become the mainstream technology of real-time application development in mobile systems based on cloud computing [1], [2]. Web services are orchestrated by service-oriented workflow applications to provide different functions. Web Service Business Process Execution Language (WS-BPEL or BPEL for short) has become the standard orchestration language. However, those applications often suffer from failures or defects, especially during the evolution of service composition. In addition, the maintenance of these applications is worried. On average, these activities account for 2/3 of the total cost of the software life cycle [3]. However, if old test cases and results can be reused, maintenance costs can be reduced to ensure that previous functionality fails due to modifications [4]. Therefore, regression testing, as one of the most important means to provide trust in software development process, has attracted more and more attention in recent years [5].

Many studies have shown that frequent regression testing is essential for successful application development [6]. However, for service-oriented workflow applications, rerunning regression test cases can be a time-consuming (day or even week) process, especially in cloud-based mobile systems. In addition, revocation of the cost of using the service (for services with access quotas or on a usage basis) or the usage of the service (for example, the stock exchange system) requires that failure can be detected as soon as the regression test cases are executed [7]. For this reason, we must resort to test case prioritization techniques to improve the testing efficiency [8]. Test case prioritization is a way of scheduling test cases to meet specific goals. The primary goal of test case prioritization is to detect faults early so that we can reduce the time and cost of maintaining service-oriented workflow applications. However, obtaining fault detection information

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

is difficult before the test is completed. Therefore, most test case prioritization approaches rely on surrogates to schedule test cases, and expect that meeting these surrogates early can lead to increased fault detection capabilities [9]. A common surrogate for test case priority is certain program entities (statements or branches) [10], [11], which is insufficient to guarantee high fault detection in some cases [12]. Therefore, we have to find a more accurate way to prioritize test cases to test service-oriented workflow applications.

References [13] and [14] have pointed out that defects in the internal structure of the software can cause software failures. Thus, the internal structure of software affects the quality of software. In addition, the study of test case priorities for service-oriented workflow applications should consider bug propagation and coverage information. Unfortunately, as far as we know, fault propagation in service-oriented workflow applications has not been fully considered in existing regression test case prioritization studies. Focusing on the modified fault propagation behavior in service-oriented workflow applications, this paper proposes a new method to schedule test cases for regression testing.

The internal structure of service-oriented workflow applications can be regarded as an activity interaction to achieve the desired goals. For example, the interaction of activities includes executing process logic, exchanging messages and calling external web services. These interactions can be analyzed by analyzing dependences between activities in a service-oriented workflow application. Changes or errors in activities will spread to other activities which are directly or indirectly dependent. In this paper, *testing importance of activity* is measured by *modification impact of activity* in a service-oriented workflow application. In addition, we can get the *test importance of test case* through its coverage information (combined with modification information). The modification information can be calculated by comparing the differences between the original version and its variants. Then, according to the importance of test cases, test cases are sorted to test service-oriented workflow applications. Before that, we need to first analyze the BPEL dependency between various activities. Apart from *control dependence*, *data dependence*, and *asyn-invocation dependence*, two other program dependences: *correlation* dependence and *synchronization dependence* are identified in this paper. Considering these dependences, we construct a graph called *BPEL activity dependence graph* to quantitatively calculate the modification impact of each activity. We experimented with our approach and 8 traditional service-oriented workflow applications based on coverage technology. The experimental results show that the test cases sorted by our method can achieve higher fault detection rate.

The preliminary version [15] of this paper has outlined the basic perspective and conducted some basic experiments to evaluate our approach. In this present paper, we systematically detail the approach based on modification impact analysis. In addition, we conducted extensive experiments as well as more data analysis to validate the observations.

This paper together with its preliminary version [15] mainly makes the following three contributions:

- Two novel WS-BPEL activity dependences, which are, correlation dependence and synchronization dependence are proposed.
- We propose the first work to quantitative measure the modification impact with activity dependences of service-oriented workflow applications.
- We propose the first work from the perspective of modification impact analysis for test case prioritization of service-oriented workflow applications.

The rest of this article is organized as follows. The II section introduces some preliminary situations. The III section shows an example to motivate our method. The IV section introduces our methods. The V section uses our approach to report the experiment. The VI section discusses our method. The VII section reviews the relevant research results. The VIII section concludes our work.

II. PRELIMINARIES

This paper discusses the test case prioritization of regression testing. Here we take BPEL workflow application as an example to illustrate our approach. BPEL workflow application is one of the most popular service oriented workflow applications [16]. To make our approach easier for readers to understand, this section introduces the prioritization of test cases and the foundation of the WS-BPEL language.

A. TEST CASE PRIORITIZATION

Test case prioritization can improve the test efficiency in regression testing [12], [17], which through designing the order of test cases to run. Combined with the definition of reference [12], we give the following definitions:

Definition 1 (Regression Test Case 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 $o \in O$ *such that* $(\forall o' \in O)[f(o) \geq f(o')]$.

The purpose of this paper is to find the order o to improve the fault detection rate. In order to quantify the fault detection rate for a given test set, f is always the metric of *weighted Average Percentage of Faults Detected* (*APFD*), *average Relative Position* (*RP*), and *Harmonic Mean of rate of Fault Deception* (*HMFD*) [18]. All of these measures range from 0 to 1, and a higher *APFD* value indicates a higher failure detection rate.

In this paper, the function f maps every permutation O to the *APFD*, *RP*, or *HMFD* value of o. 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 T that reveals fault i. The following equation gives the *APFD* value for ordering o.

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

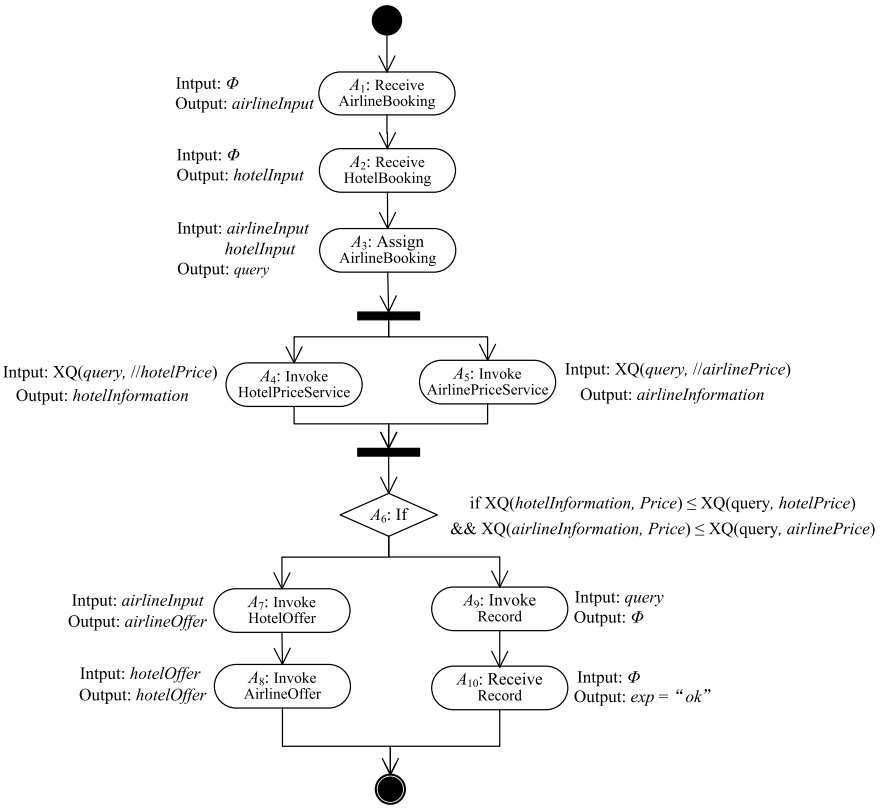


FIGURE 1. A running example: the travel agency application.

The *HMFD* value of o is computed as follows:

$$HMFD = \frac{m}{\frac{1}{TF_1} + \frac{1}{TF_2} + \cdots + \frac{1}{TF_m}} \tag{2}$$

Let $P(TF_i, i)$ be the probability that the first fault test case caused by the fault i is in the position TF_i. The RP value of fault i is computed as follows:

$$RP(i) = \frac{\sum_{TF_i=1}^{n} TF_i * P(TF_i, i)}{n} \tag{3}$$

The fault detection rate of regression test case prioritization can also be measured by other metrics [12]. Due to space constraints, we will report such results in future work.

B. BASICS OF WS-BPEL LANGUAGE

WS-BPEL combines Web services into BPEL workflow applications, which is a new service [16]. It is a language and the latest standard is version 2.0. WS-BPEL defines process logic by dividing it into basic and structured activities. Basic activities describe the basic steps of the process behavior, which include <assign>, <reply>, <receive>, and <invoke>. Structured activities represent the logic of an application, which include <while>, <flow>, <pick>, <sequence>, <if>, and <switch>. Two communication mechanisms are provided by WS-BPEL language: synchronous (request-response <invocation>) and asynchronous (one-way <invocation>). Concurrency mechanism is described by activity < flow >. Activities (basic activity) contained in the <flow> activity can be executed in any order. Furthermore, activity <link> represents the synchronization dependency between two activities in WS-BPEL language.

III. RUNNING EXAMPLE

In this section, a well-known and well-designed application, *travel agency application*, adapted from *Travel* [15], [19], is used as a motivating example to inspire our approach. First of all, we briefly outline this application.

For visual expression, the UML activity graph is used to establish BPEL codes to describe the application. The *travel agency application* is depicted in Figure 1. In this activity graph, the edge shows transformation between the two activities, while the node shows BPEL activity.

The information extraction of BPEL application (XPath Query, input or output parameters) is also used to annotate nodes [11], [20] in this paper. To ease of expression, the nodes in this application is named as A_1, A_2, ..., A_{10}. A detailed description of the *travel agency* application is as follows. Upon receiving the customer's order information (activities A_1 and A_2), the travel agent application calls both the *AirlinePriceService* (activity A_4) and *HotelPriceService* (activity A_5) to query the airline's and hotel's price. If the hotel's and airline's prices are less than or equal to the input price, this BPEL workflow application will book a hotel and

an airline ticket (activities A_7 and A_8). Otherwise, the client service failure reservation information is recorded by *RecordService*.

As we mentioned earlier, this version is modified from its initial version. Suppose that activities A_7 and A_9 have been revised, which are compared with its last version. We will let common methods (based on statement coverage or based on branch coverage) applied to the scene. Due to the coverage information of the test cases corresponding activities A_7 and A_9 is same, two test cases are randomly arranged.

According to our observation, the implementation of A_{10} is affected by the activity of A_9, and the implementation of A_8 is not affected by the activity of A_7. Therefore, the capacity of bug propagation of A_7 is weaker than that of A_9. Thus, the test cases corresponding to activity A_9 should be rerun in advance than test cases corresponding to activity A_7. In our method, we also found that the modification impacts of activities A_9 and A_7 are different, and activity A_7 is smaller. In this article, we propose such an approach to sort test cases precisely for BPEL workflow applications.

IV. OUR TEST CASE PRIORITIZATION APPROACH

In this section, an accurate test case prioritization method for BPEL workflow applications is presented. This method is based on the analysis of dependences between activities. Given a workflow application and its test suite T, our goal is to prioritize the test cases of T to find bugs as early as possible. In our approach, we sort test cases according to the testing importance of each test case. Before describing our approach, we first give a definition of the importance of test cases and activities.

A. TESTING IMPORTANCE OF ACTIVITY

In order to meet the growing needs of users, it is unavoidable that service-oriented workflow applications must be constantly developed and some modifications must be made. Therefore, software maintenance is crucial throughout the software lifecycle to reduce the potential for new bugs. Most existing approaches for solving test case priority problems are based on coverage information [11], [20]. However, as pointed out in [21], as the complexity of software systems increases, activity dependence of software is becoming more and more important in affecting software quality. However, these technologies [11], [20] do not take into account the internal structure of software. Modification impact analysis of software internal structures is important because it can also help determine the consequences of modifications [22]. Changes in the internal structure (activity) of software can affect the quality of software, which is very important in software maintenance [15]. Program tracking and module analysis techniques can be used to identify changes in the internal structure of software. Therefore, we use module dependency technology to quantitatively analyze the internal structure changes in service-oriented workflow applications. In a service-oriented workflow application, when an activity (whether or not it introduces bugs/errors) is modified, its modification is most likely to have a direct impact on its child activities. Based on this idea, in this article, we preferentially select test cases that correspond to the activities that have the greatest impact to rerun. The modification effect of this activity is mainly achieved by analyzing the dependency relationship between any two activities. The following two assumptions are the base of our approach. Informally:

Assumption 1: An error or modification in an activity will spread to other sub activities that are directly or indirectly dependent on it.

Let us take the BPEL workflow application of *travel agency* in Figure1 for example to illustrate Assumption 1. The travel agency application calls *AirlinePriceService* (activity A_5) and *HotelPriceService* (activity A_4) to check airline and hotel prices. In fact, activities A_4 and A_5 use the variable *query* is defined by A_3. Therefore, these two activities are *true data dependent* on activity A_3 [23]. With Assumption 1 in mind, the modification or error will propagate to activities A_4 and A_5 when the activity A_3 is modified or implanted errors.

Assumption 2: Any activity that relies on other activities is 100% to be propagated when other activities are modified or implanted errors.

This assumption ensures that the probability (i.e., a bug arises in an activity) is 100%. With Assumption 2 in mind, when activity A_3 is modified or seeded in a fault, activities A_4 and A_5 are influenced and the probability is the same and still 100% in this paper.

Based on the above two assumptions, we should first analyze the various program dependences between activities in service-oriented workflow applications. In addition to *data dependence*, *control dependence* and *asynchronous dependence*, we also found two other program dependences in service-oriented workflow applications: *synchronization dependence* and *correlation dependence* [15].

In this paper, we mainly consider these 5 dependences, and define BPEL activity dependence graph based on these 5 dependences. Based on BPEL program dependence graph, we can easily identify the dependences between any two activities. In order to make this paper self-sufficient, the concepts of data dependence, control dependence and aysn-invocation dependence are introduced at first.

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 (assumed that every BPEL workflow application has an entry activity) that directly controls whether A_j is executed. Taking the BPEL workflow application in Figure 1 as an example, all of the activities of $A_1 \sim A_6$ in the BPEL workflow application of Figure 1 are control dependent on entry activity.

1) CONTROL DEPENDENCE

Each BPEL workflow application is assumed to have an entry activity. For an activity A_i and an activity A_j, if the activity A_i is a predicate of a conditional branch or entry activity, and if the execution of the activity A_j is directly controlled

by A_i, then the activity A_j is called control dependent on the activity A_i. For example, as shown in Figure 1, all of the activities A_1 through A_6 in its BPEL workflow application are control dependent on its entry activity.

2) DATA DEPENDENCE

Data dependence can be divided into *true dependence*, *anti-dependence* and o*utput dependence*. More vividly, it can also be called *def-use dependence*, *use-def dependence* and *use-use dependence*. For an activity A_i and an activity A_j, if a variable is defined at A_i and used at A_j, then the activity A_j is *def-use dependent* on the activity A_i; if a variable is used at A_i and defined at A_j, then the activity A_j is *use-def dependent* on the activity A_i; if a variable is used at A_i and is also used at A_j, then the activity A_j is *use-use dependent* on the activity A_i. E.g., in Figure1, the variable *hotelInput* defined at A_2 is used at A_3, the activity A_3 is def-use dependent on the activity A_2, and there is no use-def dependence and use-use dependence in Figure1. Actually, we can avoid use-def dependence and use-use dependence by renaming variables. Therefore, when this paper refers to data dependence, it means def-use dependence.

3) ASYNC-INVOCATION DEPENDENCE

Async-invocation dependence originated from async-communication mechanisms. For an activity A_i and an activity A_j, if A_i is an ongoing one-way < invoke > activity and A_j is a <receive> activity for receiving the corresponding returned result, then the activity A_j is async-invocation dependent on activity A_i. E.g., in Figure1, A_9 is an ongoing one-way < invoke > activity and A_{10} is a <receive> activity which is responsible for receiving the corresponding returned result, so that the activity A_{10} is async-invocation dependent on activity A_9.

In this paper, we focus on these three dependences described above. In addition, we have defined two other BPEL program dependences.

The first program dependence is proposed based on the correlation mechanism between *reception* activities in a BPEL workflow application. Receive activities include <receive>, <invoke> and <onMessage>, and its two properties "*createInstance*" and "*correlation*" play key role in message routing (see WS-BPEL 2.0). If the "creatinstance" attribute value of a start activity is "yes", then this is a reception activity. Suppose there is a BPEL workflow application whose startup activity is a receiving activity with a createInstance attribute value of "yes". We can create an instance and then route all other messages received by its corresponding receiving activity to the newly created one. We found that in this BPEL workflow application, there is no such three dependences between the <receive> activity and other receiving activities that have the same correlation set as the <receive> activity. However, if the order of execution of these two activities is broken, the workflow application may deadlock, indicating that there are some dependences between two activities. We can informally define this dependency as follows:

Definition 2 (Correlation Dependence): For an activity A_i and an activity A_j in a BPEL workflow application, start activity A_i is a reception activity with a createInstance attribute value of "ye", A_j is a reception activity, both of them share the common attribute, then activity A_j is correlation dependent on activity A_i.

In the BPEL workflow application of Figure 1, the start activity A1 is a receive activity with the createInstance property set to "*yes*", and the activities $A_2, A_4, A_5, A_7 \sim A_{10}$ are receive activities. At the same time, in order to ensure that the receiving activities $A_1, A_2, A_4, A_5, A_7 \sim A_{10}$ interact with the same client, each of their messages is associated via an correlation set *ID*. Hence, activities $A_2, A_4, A_5, A_7 \sim A_{10}$ are dependent on A_1.

The second program dependence is proposed based on the association mechanism between the activities contained in the <flow> activity. The <flow> activity is a structured activity that describes the mechanism of concurrency and synchronization. There is a group of activities in the < flow > activity. Except for the defined precedence relationship of <sequence>, this group of activities can be executed in any order, and only all activities contained in the <flow> activity are executed, and the <flow> activity is finished. In the WS-BPEL language, <link> represents the synchronization dependency between activities. In the <flow> activity, if the execution order of two activities connected through < link > is reversed, the workflow application will lock up, which indicates that two activities do have some dependency. We can define this dependence informally as follows:

Definition 3 (Synchronization Dependence): For the activities A_i and A_j contained in the <flow> activity, if the activity A_j is the target of the <link> with the activity Ai as the source, the activity A_j is synchronization dependent on the activity A_i.

In the BPEL workflow application of Figure 1, we find that the execution order of activity A_4 and activity A_5 contained in the <flow> activity can be reversed at will, and there is no dependence between them. So there is no synchronization dependence in this BPEL workflow application.

So far, we have found five dependences in the BPEL workflow application, and now we can define BPEL activity dependence graph based on these five dependences. In the BPEL activity dependence graph, nodes are used to represent basic activities or structured activities (except for the Entry node). The directed line segments represent the dependences between nodes and nodes. The directed line segments that control dependences and data dependences are marked with *True* (T) or *False* (F), for the sake of simplicity, data dependence tags are generally omitted. We can informally define BPEL activity dependence graphs as follows:

Definition 4 (BPEL Activity Dependence Graph): For directed graphs <N, E>, where:

a) N is a set of n nodes containing 1 entry node and n-1 basic active or structured active nodes;

b) E is a set of directed line segments of n×n order, and the directed line segment <X, Y>∈ E from node X to node Y represents the dependence between the two activities X and Y.

The directed graph <N, E> is called the BPEL activity dependence graph.

Let us discuss how to identify all the BPEL program dependences in a BPEL process. Control dependences can be obtained by the post-dominator analysis based on the BPEL control flow graph (CFG). Data dependences in a BPEL process can be obtained based on the well-established data flow analysis techniques.

As a first-class citizen of BPEL specification, a <link> element represents a synchronization dependence directly. Therefore, the synchronization dependences introduced by <link>s can be conveniently obtained for free. Similar to the control dependence analysis, asyn-invocation dependences and interaction dependences in a BPEL process can also be obtained by traversing the BPEL CFG. However, the procedures of analyzing asyn-invocation, correlation and synchronization dependences are somewhat technical. Some syntactical details of BPEL and WSDL need to get involved here. For this reason, we assume that readers are familiar with both BPEL and WSDL. For asyn-invocation and interaction dependence analysis, the attributes ''partnerLink'', ''portType'', and ''operation'' of communicating activities are indispensable. To this end, we associate these attributes with the corresponding activity nodes in the BPEL CFG.

Let us first discuss how to identify the asyn-invocation dependences in the BPEL CFG. A <receive> activity A_j is asyn-invocation dependent on an <invoke> activity A_i if the following conditions are met: (1) A_j is reachable from A_i in the BPEL CFG. (2) A_i and A_j share the same ''partnerLink'' of a ''partnerLinkType'', where the ''portType''s of both activities are declared respectively in two ''role''s of the ''partnerLinkType''.

In addition, the synchronization and correlation dependences can be obtained from their respective definition. Now the travel agency application case of Figure 1 can be converted into the BPEL activity dependence graph shown in Figure 2. As shown in the legend in Figure 2, the dependence type is distinguished by the arrow color. This case involves the four dependences introduced earlier without synchronization dependence.

After obtaining the activity dependence graph of the BPEL workflow application, we can then analyze the impact on the entire application when an activity in the program is modified or an error occurs based on the BPEL activity dependence graph.

The method adopted in this paper is the program slicing method, which is to delete other activities in the program that have no direct or indirect dependence on the activities we analyze. For the activity dependence graph G of BPEL workflow application program and the activity Ai therein, the slicing of activity Ai is only to retain the activities directly or indirectly related to Ai in G, and delete irrelevant activities. The slicing algorithm is shown in Figure 3. The slicing

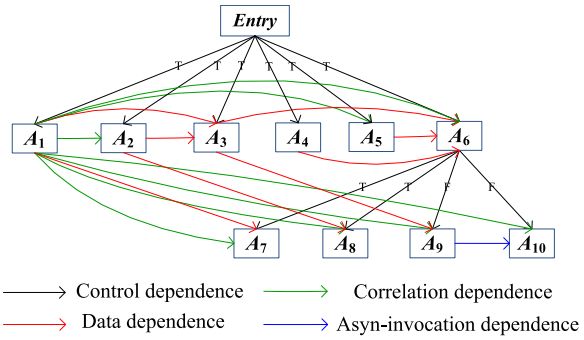


FIGURE 2. BPEL activity dependence graph for the travel agency application.

```
ForwardSlice(node m)
{
Set ddslice = Null
If node m is not traversed
   Mark node m as traversed
For each outgoing dependency edge
   Add node n to the ddslice
   for each such node m do dependencySlice(node n)
}
```

FIGURE 3. Slicing algorithm for dependence graph.

method can be divided into forward slicing and backward slicing according to different search directions. Because this paper mainly discusses the influence of activity modification or error on the application program, so forward slicing is used here. We can define Modification Impact of Activity informally as follows:

Definition 5 (Modification Impact of Activity, MIA): For the activity dependence graph G of the BPEL workflow application and its activity A_i, we call the forward slice of A_i in the activity dependence graph G as the modification impact of activity A_i, and its expression can be written as:

$$MIA(A_i) = ForwardSlice(G, A_i) \tag{4}$$

On the basis of *MIA*, we also proposed the concept of *Testing Importance of Activity* (*TIA*), that is, the sum of the number of activities affected by the modification of activity A_i in the BPEL workflow application, which is defined as follows:

Definition 6 (Testing Importance of Activity, TIA): For the activity dependence graph G and its activities A_i of the BPEL workflow application, we call the total number of activities in the MIA (A_i) of the BPEL program dependency graph G as the test importance of activity A_i, Its expression is as follows:

$$TIA(A_i) = \sum_{M} (MIA(A_i)) \tag{5}$$

According to Definition 5 and Definition 6, we can obtain the *MIA* and *TIA* of the BPEL activity dependence graph in Figure 2. Taking activity A_6 as an example, we can get

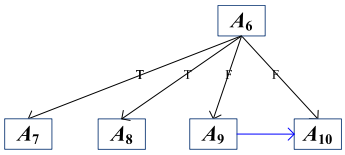


FIGURE 4. *MIA* (A_6) of Figure 2.

$MIA(A_6)$ as shown in Figure 4 by forward slicing activity A_6, then $TIA(A_6) = 5$.

B. OUR PRIORITIZATION APPROACH

After the activities in the BPEL workflow application are modified, in order to verify the impact of activity modification on the program, it is necessary to introduce active test cases for verification. When multiple activities are modified at the same time, it is necessary to determine the execution sequence of different test cases, that is, to determine the priority of test cases. To solve this problem, this section introduces the concept of *Testing importance of Test Case* (*TITC*).

Definition 7 (Test Importance of Test Case, TITC): For activity A_i in BPEL workflow application, let t_i be the test case of activity A_i, then the test importance of test case t_i is equal to the sum of TIA values of all activities in the slice diagram of MIA (A_i), and its expression is:

$$TITC(t_i) = \sum_{j=1}^{m} TIA(A_j) \tag{6}$$

A_j is the set of all activities in the $MIA(A_i)$ slice graph, $j = \{1,\ldots, m\}$, and $m = TIA(A_i)$.

According to the *TITC* value of the test case, the test case priority of the BPEL workflow application can be obtained. In summary, we summarize the test case prioritization methods as follows (see the flowchart in Figure 5):

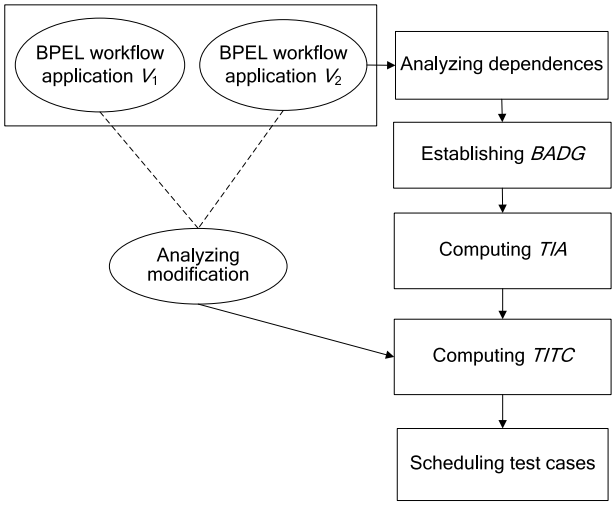


FIGURE 5. The framework of our approach.

Let the initial BPEL workflow application be P_1, and the modified BPEL workflow application be P_2.

Step 1: According to the definitions and determination methods of the various dependence relationships given above, analyze the various dependence relationships of activities in P_1 / P_2 to establish the BPEL activity dependence graph (BADG) of P_1 / P_2;

Step 2: Compare P_1 and P_2, and determine Modify Activity Set $[A_i]$ ($i = 1, \ldots, n$);

Step 3: Analyze the *Modification Impact of Activity* $MIA([Ai])$ of the Modification Activity Set $[A_i]$;

Step 4: Calculate the *Testing Importance of Activity* $TIA([A_i])$ of the Modification Activity Set $[A_i]$;

Step 5: Select the appropriate test case $[t_i]$ of Modify Active Set $[A_i]$, and use the $MIA([A_i])$ and $TIA([A_i])$ obtained in step 3 and step 4 to calculate the *Testing importance of Test Case* $TITC([t_i])$ of the test case $[t_i]$. The TITCs of the test cases are arranged in descending order. The higher the *TITC* value, the higher the priority. The test case execution order can be randomly arranged when the *TITC* values are equal.

So far, we have introduced the test sequence of the test cases for regression testing. Take the application in Figure 1 as an example. The BPEL activity dependence graph of the program has been given in Figure 2. Now we assume that the activities A_1 and A_6 in the program have been revised, then the Modify Activity Set is $[A_1, A_6]$, $MIA(A_6)$ is shown in Figure 4.

According to definition 7, the *Testing importance of test case* t_6 of A_6 is $TITC(t_6) = TIA(A_6) + TIA(A_7) + TIA\ (A_8) + TIA\ (A_9) + TIA\ (A_{10}) = 10$, the same can be obtained for the testing importance of test case t_1 of A_1 is $TITC(t_1) = 1$. Therefore, the priority of test case t_6 is higher than t_1, and t_6 should be re-run in advance.

V. EXPERIMENTS

In this part, we will test the proposed method and several typical methods through experiments. Compare and analyze the three indicators of test case priority: Average of the Percentage of Faults Detected (*APFD*), the average relative position (*RP*), and The harmonic mean of the rate of fault detection (*HMFD*), so as to illustrate the superiority of the proposed method.

A. EXPERIMENTAL SETUP

As shown in Table 1, we selected 8 BPEL applications of *A-H* to evaluate the superiority of the proposed method. These programs were selected from BPEL engines such as ActiveBPEL, Web Services Innovation Framework, IBM BPWS4J, or BPEL specifications. Table 1 details the name, source, number of XML elements, number of codes (locs), whether the correlation set has been used in the program (COR) and whether the link has been used in the program (link) of each BPEL application. It should be noted that our experiment is based on the ActiveBPEL (version 4.1) engine to run BPEL workflow applications and their partner web services.

TABLE 1. 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

Then we plant some different modifications or errors into these eight BPEL workflow applications to form different program versions, and only one modification or error is implanted into each version, which can ensure the independence of the test. Based on the idea of mutation testing, we embed errors into BPEL, WSDL and XPath respectively. In BPEL, errors are mainly logic errors or spelling errors related to execution results. In WSDL, errors are mainly embedded in various elements, including errors in types, messages, port types, bindings and services. In XPath, errors are mainly embedded in path expressions error, which will result in unable to select content or wrong content. In accordance with the above rules, we invited research partners (non paper authors) to generate a total of 59 different program versions for the eight BPEL workflow applications, and the number of versions for each BPEL workflow application is shown in the ''Versions'' column in Table 1. Similar methods are also used in [10], [11], [20] to generate different program versions.

This paper uses the test case generation method proposed in [20] to generate the required test cases. For example, as shown in Figure 6 is the program section of Ref. C gymlocker in Table 1, a test case (*locklockerinfo, unlocklockerinfo*) can be obtained by using this method to effectively test Ref. C gymlocker program. Now we build a test pool for each application of *A*-*H*, and each test pool contains 1000 test cases. We also use the construction method of test set in reference [10] to ensure that the efficiency of fault detection is not affected by the order of test case generation, that is, for application x, set the active branch coverage of test case contained in its test set *TS* (initially empty) as *AC*, and we take test case *TC* from the test pool *TP* of the program one by one, if *TC* can expand the active branch Coverage *AC*, then *TC* is added to *TS* until all activities of program *X* are executed by more than 10 test cases. Finally, we get 100 test sets for each application. Table 2 shows the statistics of the number of test cases in the test sets of 8 applications: maximum, minimum and average.

For different program versions, if more than 20% of the test set can detect errors in this version, this version will be discarded because these faults are basically eliminated during developer development and before regression testing.

```
<bpws:variables>
    <bpws:variable messageType="tns:lockerInfoMessage"
name="lockLockerInfo"/>
    <bpws:variable messageType="tns:unlockerInfoMessage"
name="unlockLockerInfo"/>
    <bpws:variable messageType="tns:statusMessage"
name="status"/>
  </bpws:variables>
……
<bpws:sequence>
    <bpws:receive createInstance="yes" name="unlockerReceive"
operation="unlock" partnerLink="user"
portType="tns:gymLockerPT" variable="unlockLockerInfo">
    </bpws:receive>
    <bpws:assign>
      <bpws:copy>
        <bpws:from expression="'ok'"/>
        <bpws:to part="information" variable="status"/>
      </bpws:copy>
    </bpws:assign>
    <bpws:receive name="lockerReceive" operation="lock"
partnerLink="user" portType="tns:gymLockerPT"
variable="lockLockerInfo">
    </bpws:receive>
    <bpws:reply name="lockerReply" operation="lock"
partnerLink="user" portType="tns:gymLockerPT"
variable="status"/>
    <bpws:reply name="unlcokerReply" operation="unlock"
partnerLink="user" portType="tns:gymLockerPT"
variable="unlockLockerInfo"/>
</bpws:sequence>
```

FIGURE 6. BPEL program segment of GymLocker.

TABLE 2. Statistics of test suite sizes.

	A	B	C	D	E	F	G	H	Mean
Max.	132	104	118	132	134	159	125	107	126.4
Mean	68	46	58	70	75	82	66	57	65.3
Min.	25	16	19	21	30	27	24	22	23.0

Five different test case priority technologies are discussed in the test of this paper. the approaches of total activity coverage prioritization and additional activity coverage prioritization in our experiments are the technologies of Mei et al [20].

1. Random ordering. The order of test cases in test set is arranged by computer pseudo randomly;

2. Total activity coverage prioritization [20]. It is sorted by the total number of activities covered by each test case in the test set. Test cases with equal total activity coverage are randomly sorted;

3. Additional activity coverage prioritization [20]. Which is to say, after the execution of the test case, the increase of total activity coverage is the largest among all the test cases being executed. When the total activity coverage reaches 100%, set zero and continue to sequence iteratively according to this rule. Test cases with equal increase of activity coverage are randomly selected;

4. Modification impact analysis based approach. The activity modification impact analysis method proposed in this paper, which is ranked according to calculated test case test importance. Test cases with equal importance are randomly ordered;

5. Optimal ordering. When the error location and type are known, the test cases are optimally arranged to make the test set the most efficient in discovering the errors in the program. This method is the theoretical upper limit of test case prioritization.

For each subject and each constructed test suite T for the subject, we applied every technique to prioritize T. We executed each prioritized T against every modified version of the subject. It used the outputs of the original version as expected outputs. It calculated the corresponding *APFD*, *PR*, and *HMFD* values. In total, 783,451 *APFD* values, 476 *RP* values, and 783,451 *HMFD* values were collected.

B. RESULTS AND ANALYSIS

For each application and test set Ts of each application, the priority of test cases in test set Ts is determined through Random ordering, Total Activity Coverage Prioritization, Additional Activity Coverage Prioritization, Modification impact analysis based approach and Optimal ordering. Each modified version of each program executes the test set Ts according to different test case priorities. The output value of the original version is used as the expected output value. The above methods are compared by statistical analysis of the values of *APFD*, *RP* and *HMFD*. In this experiment, we use each test set Ts to repeat the test 10 times, and find the average value of each test as the output result. In the end we obtained a total of 783451 *APFD* values, 476 *RP* values and 783451 *HMFD* values.

Figure 7 is a box-whisker diagram of *APFD* values. Figure 7 (a) - Figure 7 (h) respectively show the *APFD* measurements of Ref. A - Ref. H, the eight applications applying five test case prioritization techniques. The vertical axis of each sub-picture represents the *APFD* value, and the horizontal axis represents different test case prioritization techniques (Random indicates Random ordering; Total-activity indicates Total Activity Coverage Prioritization; Addtl-activity indicates Additional Activity Coverage Prioritization; Modifi-impact means Modification impact analysis based approach; Optimal means Optimal ordering, the same below).

It can be seen from Figure 7 that the *APFD* values of three methods of Total-activity, Addtl-activity, and Modifi-impact are higher than random ordering. For these three methods, although the *APFD* value is almost the same sometimes (such as Ref.C & Ref.H), in most cases, the *APFD* value of Modifi-impact is better than the other two methods. Therefore, through this experiment, we can find that the approach of Modifi-impact outperforms than approaches of Total-activity and Addtl-activity.

We also collected the 1/4-quantile, median, 3/4-quantile and average of *APFD* values, *RP* average and *HMFD* average of five test case prioritization techniques. Figure 8 shows the results. Each sub graph shows the 1/4 quantile value, the median value and the 3/4 quantile value of different statistical values in the form of box-plot. For example, the fourth box in Figure8(b) represents the 1/4-quantile value, the median value, and the3/4-quantile value of the *APFD* median value of the Modifi-impact method.

We find across three subfigures (Figure 8 (a) – (c)) that, for each technique, as the percentile progresses (namely, from 25th percentile to median and from median to 75th percentile), the performance of the technique shows a trend. As is depicted in Figure 8 (d), the box-plot of Random and Optimal show the worst and the best mean *APFD* in the figure. Our approach (Modifi-Impact) outperforms approaches of Total-Activitiy and Addtl-Activity. This result is consistent to the result of Figure 7. The bars for measurement *RP* and *HMFD* in Figure 8 (e) and Figure 8 (f) also demonstrate this similar result.

C. THREATS TO VALIDITY

This paper proposes a new way to improve the method of test case prioritization for BPEL workflow applications. The proposed method based on modification impact analysis is only for the initial application of the improved method. In this paper, the number of BPEL workflow application samples and the number of modified versions are limited, which can not cover all cases, and our experiment only runs in ActiveBPEL (version 4.1) engine. In view of the above situation, we will continue to apply this method to a variety of large-scale BPEL applications, and further improve the method through practice. At the same time, in the future, we will test in different engines to reduce platform threats.

At the same time, there are three main reference indicators for discussing the superiority of the program test case prioritization method: *APFD*, *RP*, and *HMFD*. Because these three values need to be obtained when the error is known, they cannot be obtained until the test is complete. However, these three reference indicators can verify the superiority of the prioritization method after the test being completed.

VI. DISCUSSION

In the research of this article, we have proved that in some aspects, our proposed method based on modified impact

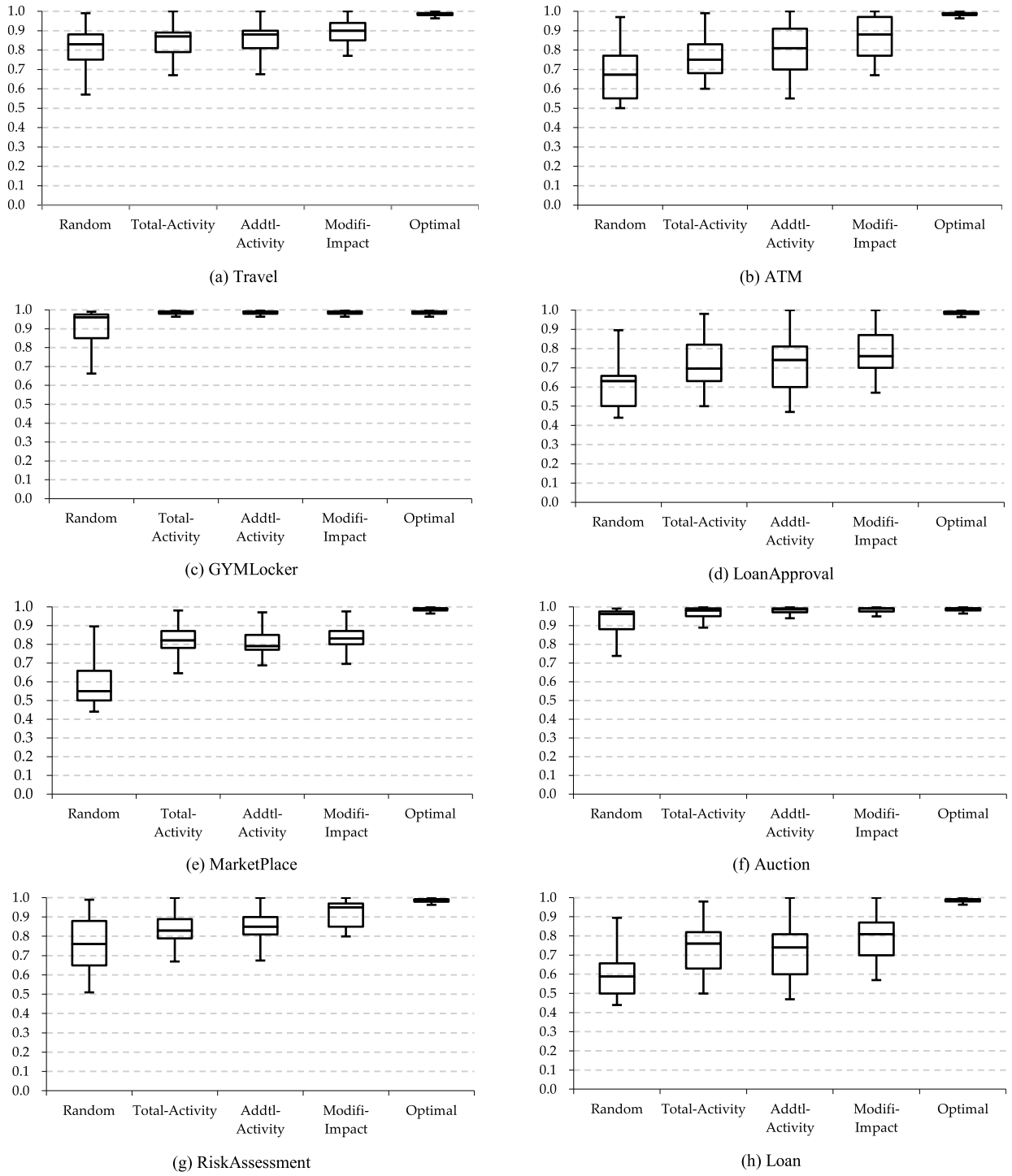


FIGURE 7. Comparisons on each BPEL workflow application using APFD measurement.

analysis is better than some traditional methods, which can efficiently test BPEL workflow applications, but there are still some problems and disadvantages:

Firstly, because the method proposed in this paper needs to obtain and analyze the BPEL activity dependence graph,it will add some additional workload. Especially for large BPEL workflow applications, it will add more workload. In this article, we quote some relatively simple BPEL workflow applications to test the effectiveness of our proposed method, and the results also show that our improved method is a new way worth trying. In the following exploration, we need to further optimize the algorithm to achieve a trade-off between program detection effectiveness and work efficiency [37].

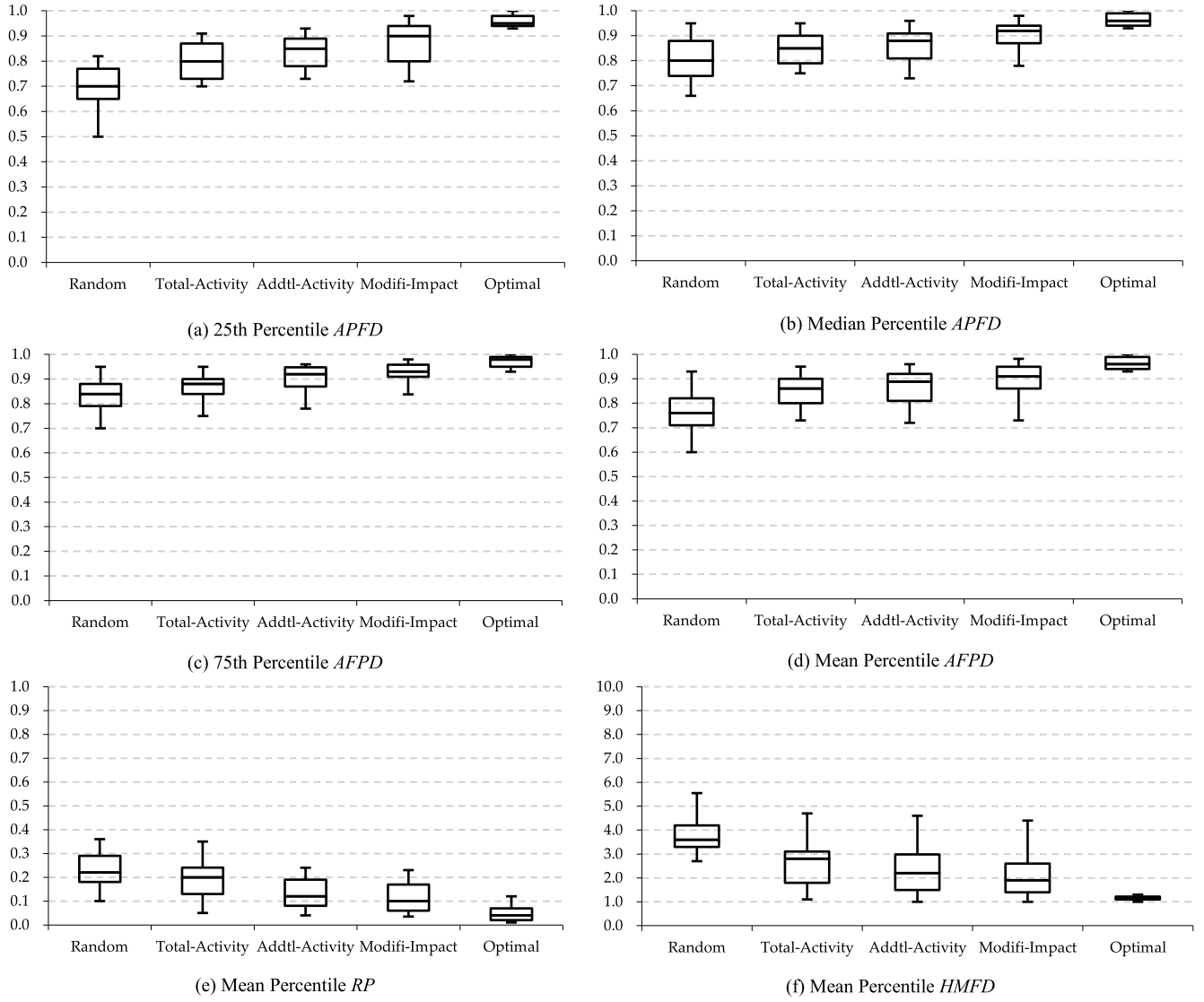


FIGURE 8. Overall comparisons in terms of APFD, RP, HMFD.

Secondly, in the quantitative calculation of test case importance, we assume that all activities can be modified or affected; and assume that errors or modifications of an activity will be 100% propagated to all activities that directly and indirectly depend on this activity in the activity dependence graph. These two assumptions may be inconsistent with the facts, and this is the problem that we will solve next.

Thirdly, our current research ignores some of the advanced activities of WS-BPEL 2.0 and only considers its basic elements and structure. For example, the fault handling <scope> activity used to eliminate <scope> faults in WS-BPEL 2.0 has not been considered. These are also issues that we need to consider in the future.

In addition, the test case prioritization method for service-oriented workflow applications proposed in this article is not limited to workflow applications that are applied to BPEL, but can also be applied to applications such as executable WS-CDL applications, OWL-S Applications, flow-driven Mashup applications, and other service-oriented workflow applications.

VII. RELATED WORK

Regression testing is a recognized and effective testing technology to confirm that the modified program has not introduced errors or caused other code to run improperly. The research on regression testing mainly focuses on test case generation, test method selection, test prioritization, and test minimization. Scientists have proposed various methods for the above problems [8], [9], [12]. In this section we first review the work on test case generation methods and test method selection. These are the basis for our prioritization of test cases. We will then review the prioritization of regression test cases and compare these methods with our proposed method based on Modification impact analysis.

A. TEST CASE GENERATION OF SERVICE-ORIENTED WORKFLOW APPLICATIONS

Mei *et al.* [27] addressed 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.* [27] proposed 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. By using this model, they also studied the test case prioritization for regression testing of WS-BPEL applications. Ni *et al.* [28] generated a message sequence based on Message Sequence Graph (MSG), which contains the information of message sequences and message parameters. This approach mainly considers correlation mechanism and generates message sequence based on random testing. However, this work may still instantiate some idle instances. Moreover, Tan *et al.* are focusing on how to design, analyze, and deploy web service–based workflows for both business and scientific applications [29]–[31].

For this reason, we propose a method based on satisfiability module theory solver in reference [19], which is a message sequence generated on a structured basis, and uses concurrent BPEL activity path coverage standard to sample the execution of WS-BPEL. Because DPE and association mechanism are considered, this method will not instantiate any idle instances. However, reference [28] is more practical for some applications without associated activities and fewer predicates, which belongs to a simplified method.

B. REGRESSION TEST SELCTION OF SERVICE-ORIENTED WORKFLOW APPLICATIONS

These methods [32], [33] about regression test selection in BPEL workflow application are mostly variations of the approaches proposed by Rothermel and Harrold [34]. These methods are in view of the control flow analysis between the old version. and the new version. Based on the modified and original control flow graph, the modified nodes or edges are identified by graph comparison algorithm. Test cases that override changing nodes or edges are selected as affected cases and selected for rerun. These methods are usually different for different test objectives and test perspectives of BPEL applications. From the service integrator's perspective, capture the impact of process changes, binding changes, and interface changes, Li *et al.* proposes a regression test selection method based on extensible BPEL flow graph (XBFG) [32]. The XBFG path generated from partner service and BPEL workflow is divided into two parts. One part is retested by selecting test cases in the original version, and the other part is tested by generating new test cases after executing message sequence comparison, path condition analysis, and XBFG path comparison. From the point of view of service integrator, uses a special algebraic expression is selected by Lin *et al.* [33] to represent test paths, and compares these test paths to select the affected test cases. For this method, the influence of concurrency on test path can be divided into two categories. By applying impact analysis rules to analyze the above two types of concurrent control processes, this method can not only select test cases through the affected path, but also select test cases without test path.

Considering the unique features of BPEL applications, we present an optimal controller guided by behavioral difference based on BPEL program dependence graphs [35]. Compared with the previous approaches, our approach can eliminate some unnecessary test cases to be rerun. We show the validity and feasibility of our approach through empirical studies.

C. TEST CASE PRIORITIZATION OF SERVICE-ORIENTED WORKFLOW APPLICATIONS

There are two types to solve the prioritization problems of BPEL workflow application test cases. The first type supposes that the test environment is static. Based on this, the partner web services that BPEL workflow application calls are unchanged for regression testing. Mei *et al.* [10], [20] mainly study whether XPath and WSDL information contribute to effective regression testing of service-oriented workflow applications. Based on the above information, they proposed a multi-level coverage standard. This multi-level coverage standard includes XPath, WSDL, and business processes. In addition, they developed a series of test case prioritization approaches based on the model. Unlike them, we analyze the dependences between activities to calculate the importance of modification activities to test cases. With these dependences, this paper proposes the concept of Testing Importance of Activities (*TIA*) and Testing Importance of Test Case (*TITC*) to prioritize the sequence of test cases. As a matter of fact, these two methods complement each other. Our method is depth first and Mei *et al.*'s method is breadth first. By considering the quota of specific external web service requests, To reduce the cost of service invocation, Zhai *et al.* [18], [36] take advantage of the dynamic characteristics of service selection. The method proposed by us belong to this category. We propose a test case prioritization method considering the impact of activity modification, which belongs to this category. As a matter of fact, these methods and our methods are complementary.

The second type of hypothetical testing environment is unstable. In this case, the partner web services that BPEL workflow applications call can be different. Mei *et al.* [11] proposed a new strategy, which rearranges test cases in planned regression test sessions based on the changes of tested services detected during actual regression test session. They also proposed 3 special strategies that can be integrated with existing test case prioritization methods. However, their method does not consider the modification impact of internal activities in service-oriented Workflow applications.

VIII. CONCLUSION

In regression testing, test case prioritization is a method to improve the efficiency of error detection in modified service-oriented workflow applications by changing the test order of test cases. For service-oriented workflow applications, this paper proposes a new test case prioritization method considering the internal structure of their activities and fault propagation behavior. Compared with the traditional methods, the proposed method is proved to be efficient, feasible and effective.

On the other hand, the application effect of the method proposed in this paper in some large-scale service-oriented workflow applications needs to be tested, and some advanced activities of WS-BPEL 2.0 are not considered, which are all to be further studied in this paper.

REFERENCES

[1] X. Chen, J. Lin, Y. Ma, B. Lin, H. Wang, and G. Huang, "Self-adaptive resource allocation for cloud-based software services based on progressive QoS prediction model," *Sci. China Inf. Sci.*, vol. 62, no. 11, pp. 1–3, Nov. 2019.

[2] D. S. Han, "Modeling and verification approach for temporal properties of self-adaptive software dynamic processes," *J. Comput. Appl.*, vol. 28, no. 1, pp. 165–187, Nov. 2018.

[3] H. Mei, "A Review of Software Engineering Research in China," *ACM SIGSOFT Softw. Eng. Notes*, vol. 42, no. 4, Feb. 2018.

[4] Z. Hong and Z. Y. Feng, "Collaborative testing of Web services," *IEEE Trans. Services Comput.*, vol. 5, no. 1, pp. 116–130, Jan./Mar. 2012.

[5] Z. Y. Zhang, C. Zhenyu, and X. Baowen, "Research Progress on test case evolution," (in Chinese), *J. Softw.*, vol. 24, no. 4, pp. 663–674, Jan. 2013.

[6] Z. J. Ding and Z. X. Zhou, "Survey on Web service composition testing," (in Chinese), *J. Softw.*, vol. 29, no. 2, pp. 299–319, Feb. 2018.

[7] X. Chen, J. H. Chen, X. L. Ju, and Q. Gu, "Survey of test case prioritization techniques for regression testing," (in Chinese), *J. Softw.*, vol. 24, no. 8, pp. 1695–1712, Feb. 2013.

[8] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and verification in service-oriented architecture: A survey," *Softw. Test., Verification Rel.*, vol. 23, no. 4, pp. 261–313, Jan. 2013.

[9] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test., Verification Rel.*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[10] L. Mei, W. K. Chan, T. H. Tse, and R. G. Merkel, "XML-manipulating test case prioritization for XML-manipulating services," *J. Syst. Softw.*, vol. 84, no. 4, pp. 603–619, Apr. 2011.

[11] L. Mei, W. K. Chan, T. H. Tse, B. Jiang, and K. Zhai, "Preemptive regression testingof workflow-based Web services," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 740–754, Sep./Oct. 2015.

[12] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, Jan. 2018.

[13] M. Hamill and K. Goseva-Popstojanova, "Common trends in software fault and failure data," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 484–496, Jul. 2009.

[14] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Trans. Softw. Eng.*, vol. 32, no. 10, pp. 771–789, Oct. 2006.

[15] H. Wang, J. Xing, Q. Yang, D. Han, and X. Zhang, "Modification impact analysis based test case prioritization for regression testing of service-oriented workflow applications," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, pp. 39–48.

[16] *Web Services Business Process Execution Language Version 2.0: OASIS standard*, Organization for the Advancement of Structured Information Standards (OASIS), Burlington, MA, USA, 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[17] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Test generation and test prioritization for simulink models with dynamic behavior," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 919–944, Sep. 2019.

[18] K. Zhai, B. Jiang, and W. K. Chan, "Prioritizing test cases for regression testing of location-based services: Metrics, techniques, and case study," *IEEE Trans. Services Comput.*, vol. 7, no. 1, pp. 54–67, Jan./Mar. 2014.

[19] H. Wang, J. Xing, Q. Yang, W. Song, and X. Zhang, "Generating effective test cases based on satisfiability modulo theory solvers for service-oriented workflow applications," *Softw. Test., Verification Rel.*, vol. 26, no. 2, pp. 149–169, Mar. 2016.

[20] 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 Trans. Services Comput.*, vol. 8, no. 5, pp. 658–673, Sep. 2015.

[21] W. Abdelmoez, M. Shereshevsky, B. Yu, S. Bogazzi, M. Korkmaz, and A. Mili, "Quantifying software architectures: An analysis of change propagation probabilities," in *Proc. 3rd ACS/IEEE Int. Conf. Comput. Syst. Appl.*, Jan. 2005, pp. 687–694.

[22] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Manage. Sci.*, vol. 52, no. 7, pp. 1015–1030, Jul. 2006.

[23] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, Jul. 1987.

[24] J. Yan, Z. Li, Y. Yuan, W. Sun, and J. Zhang, "BPEL4WS unit testing: Test case generation using a concurrent path analysis approach," in *Proc. 17th Int. Symp. Softw. Rel. Eng.*, Nov. 2006, pp. 75–84.

[25] Y. Yuan, Z. Li, and W. Sun, "A graph-search based approach to BPEL4WS test generation," in *Proc. Int. Conf. Softw. Eng. Adv. (ICSEA)*, Oct. 2006, pp. 14–22.

[26] C.-H. Liu, S.-L. Chen, and X.-Y. Li, "A WS-BPEL based structural testing approach for Web service compositions," in *Proc. IEEE Int. Symp. Service-Oriented Syst. Eng.*, Taiwan, China, Dec. 2008, pp. 18–28.

[27] L. Mei, W. K. Chan, and T. H. Tse, "Data flow testing of service-oriented workflow applications," in *Proc. 13th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 371–380.

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

[29] K. Huang, Y. Fan, and W. Tan, "Recommendation in an evolving service ecosystem based on network prediction," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 3, pp. 906–920, Jul. 2014.

[30] W. Tan, Y. Fan, M. Zhou, and M. Zhou, "A Petri net-based method for compatibility analysis and composition of Web services in business process execution language," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 1, pp. 94–106, Jan. 2009.

[31] Y. Du, W. Tan, and M. Zhou, "Timed compatibility analysis of Web service composition: A modular approach based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 594–606, Apr. 2014.

[32] B. Li, D. Qiu, H. Leung, and D. Wang, "Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1300–1324, Jun. 2012.

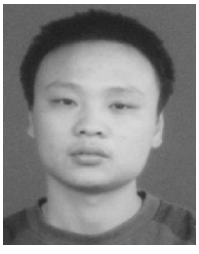
[33] F. Lin, M. Ruth, and S. Tu, "Applying safe regression test selection techniques to java Web services," in *Proc. Int. Conf. Next Gener. Web Services Practices*, Sep. 2006, pp. 133–142.

[34] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Trans. Softw. Eng. Methodol. (TOSEM)*, vol. 6, no. 2, pp. 173–210, Apr. 1997.

[35] H. Wang, J. Xing, Q. Yang, P. Wang, X. Zhang, and D. Han, "Optimal control based regression test selection for service-oriented workflow applications," *J. Syst. Softw.*, vol. 124, pp. 274–288, Feb. 2017.

[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," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2010, pp. 211–218.

[37] S. Tahvili, M. Bohlin, M. Saadatmand, S. Larsson, W. Afzal, and D. Sundmark, "Cost-benefit analysis of using dependency knowledge at integration testing," in *Proc. 17th Int. Conf. Product-Focused Softw. Process Improvement*, 2016, pp. 122–130.



HONGDA WANG received the Ph.D. degree in information system engineering from the PLA University of Science and Technology, Nanjing, in 2017. He is currently a Lecturer with Naval Logistics Academy. His research interests include service computing and workflow technology.

MAN YANG received the M.S. degree from the Hebei University of Technology. Her research interests include software engineering and service computing.

LIHUA JIANG received the Ph.D. degree in business management from the Tianjin University of Finance and Economics. She is currently a Professor with Naval Logistics Academy. Her research interests include management innovation, and logistic and supply chain management.

JIANCHUN XING (Member, IEEE) received the B.S. and M.S. degrees in electrical system and automation from the Engineering Institute of Engineering Corps, China, in 1984 and 1987, respectively, and the Ph.D. degree in information system engineering from the PLA University of Science and Technology, China, in 2006. He is currently a Professor with the Army Engineering University of PLA. His research interests include intelligent control, artificial intelligence, and information processing.

QILIANG YANG (Member, IEEE) was born in 1975. He received the Ph.D. degree from Nanjing University. He is currently an Associate Professor with the Army Engineering University of PLA. His research interests include self-adaptive software systems, mission-critical systems and software and pervasive computing, and cyber-physical systems. He is a member of CCF.

FUYONG YAN was born in 1976. He received the M.S. degree from the PLA University of Science and Technology. He is currently a Lecturer of PLA 32752 Troops. His research interests include artificial intelligence instrument, electromagnetic compatibility technology, and teaching of equipment support training.