2021 7th International Conference on Web Research (ICWR)

On-demand Test as a Web Service Process (OTaaWS Process)

Elaheh Habibi*
ICT Department
Niroo Research Institute (NRI)
Tehran, Iran
ehabibi@nri.ac.ir

Seyed-Hassan Mirian-Hosseinabadi*
Computer Engineering Department
Sharif University of Technology
Tehran, Iran
hmirian@sharif.edu

2021 7th International Conference on Web Research (ICWR) | 978-1-6654-0426-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICWR51868.2021.9443109

***Abstract*— Service-Oriented Architecture (SOA) is a software design framework where distributed services are provided and used by applications over a network. By using this architecture new challenges appeared for software testing. Lack of access to the source code and runtime binding are examples of these challenges. Functional testing of services assures the quality of service-oriented applications. In this paper, we introduce a new 3-phase consumer and provider contract-based process for web service testing to overcome SOA testing challenges. Our proposed process, namely OTaaWS, consists of Test Automation, Test Execution and Test Management steps on the provider's side, which can be done on-demand. Using this process, we have run the suggested test as a web service on a simple application called Get Committee Information using two web services. As a result, two of the total generated test cases (25 test cases) related to web service #1, are failed due to the functionality changes of this web service.**

Keywords*— Web Service, Service Oriented, Functional Test, Test as a Web Service, Online Test.

I. Introduction

The increasing complexity of software systems requires new solutions to develop systems that can overcome these complexities. One of the solutions is the use of Service-Oriented Architecture (SOA). The main reason for using this architecture is the idea of "software as a service" in which the software is designed to be used by other systems. In fact, other systems can register for a service and use it. The service has a mechanism to get access to one or more functions. Each service provides certain capabilities and has a well-defined interface through which the service can be used [1].

In current implementations of service-oriented applications, offline testing, which is done during the design phase, has several drawbacks. Undesirable consequences in executing faulty services, such as loss of money and unsatisfied users, are the examples of these drawbacks. In Contrast, online testing is performed during the operation phase of service-oriented applications [2]. When the application invokes the service instance, online test cases can be run through a service to detect defects and prevent undesired consequences on the application. To better understand, the online and the offline testing, we provide the following motivating example.

Motivating example: Assume an airplane reservation website as a client side application call three web services for gathering and displaying the airplanes' information to the user. In the offline testing the application and its web services are tested before each release in the software development cycle. So, if one of the web service failed when the application has been called them, this failure cannot detect with the offline testing and the user will see the error on the website. But, the online testing runs when the user call for the airplanes' information. Therefore, the application can detect the failure and show the proper message to the user. The response time in the online testing is reduced and the user should wait more time to see the needed information.

To tackle this problem and use the advantages of both testing, we propose a service consumer and provider testing process with the On-demand (Online/Offline) Test as a Web Service (OTaaWS).

Lack of access to the source code of the web services on the consumer side and restricted visibility at the interface level, are the main problems with the functional testing of web services [2].

Various approaches have been suggested for testing services. Most of these methods are based on the documents of services such as Web Services Description Language (WSDL) [2]. However, these documents only include the signature of inputs, outputs and some preconditions of the service functions. Therefore, the functionality of the services cannot be effectively tested. Our proposed testing process offers a test web service to solve this problem using the functional test cases that are automatically generated in the dynamic and static mode, executed and managed on the provider's side. In this process, the test data are prepared based on the requirements and input data of the consumer's application and the test cases are created from these requirements through the rule-converting engine.

More than one web service may be used in a service-oriented application. In such applications, if the services are interdependent, the error in one dependent service will lead to release in other services. Therefore, we also consider integration test cases of the services in the suggested test web service.

In summary the contribution of this paper are as follows:

- We propose a new testing process for the application with the web services (SOA application).
- We use both online and offline testing. In the online testing the test cases generated automatically in the static and dynamic mode for the specific and different application domain.

978-1-6654-0426-6/21/$31.00 ©2021 IEEE

- For creating more proper test cases, requirements of the consumer converted to the context-free grammar.
- By running OTaaWS, we can also have the regression testing.

The rest of the paper is structured as follows; In Section II, we give an overview of Service-Oriented Architecture, Web Service Testing and Software Testing Techniques. In Section III, we present the related works. Section IV introduces OTaaWS process. The OTaaWS process will be run on a simple service-oriented application. In this application, the user can get the committee information through searching the code of committee. Section V shows the results and the evaluation of results. Finally, Section VI reviews the proposed testing process and highlights the challenges and future research issues.

II. Overview

In this section, we briefly introduce the notion and concepts of service-oriented architecture and web service testing.

A. Service-Oriented Architecture

Service-Oriented Architecture is a technology of designing a software system to provide services to the end-user applications distributed in a network. In other words, the service-oriented architecture is an architectural reference model for the web services [1]. In this reference model, a SOA application uses a set of communicating components, known as services [2].

A service in SOA is a self-contained application function and a reusable business component. It can describe itself, so that other services and the applications can discover, locate and call the service through protocol messages [2]. Web services used in applications have different operational types. Researcher, processor or calculator are a variety of web services [3, 4]. In the researcher type, the web service, based on the data received from the application, finds the relevant information. In the processor type, the web service performs an operation for the application and notifies the performing results. Banking transaction in the online shopping is an example of this type. A calculator-based web service, computes calculations based on input data and returns the calculation result to the application, such as mathematical or financial calculations.

A web service can be generated in two ways: bottom-up or top-down. In the bottom-up method, the developer initially writes the code for the function of the web service, with any programming language, and then uses a tool to generate a file describing the web service. This is a simple procedure, but it is not suitable for continuous changes. On the contrary, in the top-down method, a description file of the web service is first prepared and then a tool for generating code templates is used. In this case, as long as the format of the messages sent between the service provider and the service consumer does not change, the content of a web service can be changed in both dynamic and static mode. Generating dynamic mode means changing and updating the content of the web service every time it is called and used. However, in the static generation mode, the content is customized for application and then used [5].

SOA has multiple stakeholders involved. The recent research have specified five stakeholders in SOA. These stakeholders are the developer, the provider, the integrator, the third-party certifier and the user, described as follows [3]:

- Developer: The developer is the stakeholder who implements the service and the only one that has access to the source code of the service.
- Provider: The provider is the stakeholder who deals with Quality of the Service (QoS) in the SOA. The provider ensures that the service applies all the conditions defined in the Service Level Agreement (SLA).
- Integrator (Consumer): The integrator is the stakeholder who uses services in a service composition or an application.
- Third-party certifier: The third-party certifier is the stakeholder who provides testing services to the other stakeholders.
- End-user: The end-user is the stakeholder who uses the service through an application or platform.

B. Web Service Testing

Service-centric System Testing (ScST) includes testing of the basic service functionality, service interoperability, some of the SOA functionalities, nonfunctional concepts such as QoS and load/stress testing [6]. There are many new challenges involved in testing different aspects of web services compared with testing of traditional software. These challenges include the following [6]:

- Web service testing due to the lack of source code on the application side.
- Web service testing due to the multiple stakeholders involved in SOA activities.
- Specification-based testing using some of the web service specifications such as WSDL regardless of functional aspects.
- Run-time testing of SOA activities (discovery, binding and publishing) due to the dynamic nature of service-oriented architecture.

Most of the web service testing challenges are due to the limitations that the testers typically face. The complex nature and the limitations of accessing web services are two main reasons of these challenges. The distributed nature of web services based on multiple protocols such as UDDI (Universal Description, Discovery, and Integration) and SOAP (Simple Object Access Protocol), together with the limited system information provided with specifications, makes ScST challenging [7]. There are different categories of testing techniques in different views. A functional and non-functional testing is an example of these categories [6]. In this paper, our focus is on the functional testing.

According to the definition of the test case in the IEEE standard, three elements are required: test inputs (also referred to as test data), expected results and pre/post conditions of execution. In order to generate test cases, test data must first be generated. Test data can be extracted from different sources such as source code, system requirements and use cases. In the proposed OTaaWS process, the specification of web service consumer has been considered as the source of required test data.

The main problem with software testing is the huge number of possible inputs. Therefore, regardless of different levels of testing, it is impossible to test with all inputs. Coverage criteria is the solution to overcome this problem. There are many advantages, such as improving the quality and reducing the cost of test data generation, through using coverage criteria. By definition, coverage criterion is a rule or collection of rules that created test requirements. All test coverage criteria can be apply on the inputs, on the just four mathematical structures: input domains, graphs, logic expressions, and syntax descriptions (grammars). A test requirement is a specific element of a software artifact that a test case must satisfy or cover [8].

III. Related Works

In this section, we present a survey of several existing recent works in Web Service Testing. These related works are classified in two category, Online Testing and Test as a Web Service, according to the strategies we have chosen for our OTaaWS process.

A. *Online Testing*

Online testing is performed during the operation phase of service-oriented applications. So, when the application invokes the service instance, online test cases can be run to detect defects and prevent undesired consequences on the application [9]. Hielscher et al. [9] have suggested the PROSA framework. PROSA introduces the key online testing activities and techniques to detect changes and deviations before they can lead to undesired consequences. This paper discusses how those activities can be developed by applying the PROSA on the service-oriented application, travel-planning service, as a case study. Investigating the possible impact of the execution of test cases on the performance of the application is one of the key issues of the PROSA. Wang et al. [10], have explained the motivation of the online testing, differences between the online and the offline testing, concerns and the key problems of the online testing. Increasing the performance of the online testing, improving the efficiency and supporting the reuse of design component are the main challenges. Deussen et al. [11] have presented the design and implementation of an online validation platform. In this paper, the proposed architecture is applied on the active networks by using TTCN-3 as test specification and implementation language to define test procedures for black box testing of distributed systems. Chan et al. [12] have proposed a cost-effective metamorphic approach for online services testing. The offline testing determines a set of successful test cases to construct their corresponding test cases for the online testing. These test cases will be executed by metamorphic services. They apply the proposal on the foreign exchange dealing service application. Bai et al. [13], have presented a ConfigTest as an event-based approach for detecting online changes of the test organization, test scheduling, test deployment, test case binding, and service binding. In this paper, they analyze the collaboration diagrams of various testing reconfiguration scenarios. The ConfigTest approach applies on the service-based book ordering system.

B. *Testing as a Service*

A company, Tieto, a service provider, first introduced the concept of Test as a Service (TaaS) in 2009. Testing as a service is a model for outsourcing testing responsibilities such as planning, automation, execution and management to external providers. In this model, the service consumer prepares and delivers the requirements to the provider [14]. In 2011, Oracle released a document on the TaaS. In this document, Oracle provides a cloud-based platform for automated testing of services, including load tests and functional tests, with traceability. Using the proposed method, the consumer request for the test and the test tools are implemented according to the test scenarios [15]. Leanapp Co. (now called Keylane, an insurance software service provider) has also introduced TaaS as a new testing technique and the key factor that determines the success of standard software. This white paper takes a closer look at stages in testing, regression testing and TaaS [16]. Mishra et al. [14] have presented a TaaS framework. In this paper, the implemented TaaS process and its architecture are compared with the traditional software testing process. The suggested solution also explains use of cloud technology for TaaS implementation to have an effective utilization of resources. In addition, a pricing model for test outsourcing is proposed. Data privacy, security and downtime of the provider are major limitations with TaaS. Pardeshi et al. [17] have reviewed TaaS on the cloud. This paper gives a comprehensive view on TaaS, its workflow and different types of cloud-based software testing. Scalability, performance and security testing are the major issues in cloud testing. Bai et al. [18], have suggested TaaS in cloud computing. In this paper, the Consumer’s requirements (based on the service level agreement) are collected through a web-based user interface. Then data and test cases are generated. The necessary resources for testing in the cloud environment are prepared. Finally, the run-time results that are detectable and traceable are returned to the consumer.

In addition to studies, different tools have been developed for web services testing. SoapUI is one of the most popular open source web service testing tool. In these tools, test requirements and test data are manually provided as test cases and are replaced in ready-to-run templates for web service testing. Our Web Testing Service also works like a tool. With the difference that test data and test cases are automatically generated from the user's requirements through a grammar-based coverage.

IV. OTaaWS Process

The On-demand Test as a Web Service (OTaaWS) is based on the service provider and consumer contract-based architecture (Fig. 1). In this process, the consumer (Integrator) is the owner of the application composed of web services and has a request for OTaaWS to test the functionality of the application and its services. The provider of OTaaWS responds to the request with a test plan. After the agreements, the provider should generate test cases and execute them through a test web service and finally manages test results. In this proposed process, we consider that the application under the test uses one or more web services. With OTaaWS, the consumer is able to request the regression testing [7]. Note that the provider of OTaaWS is different from the providers of web services used in the service-oriented application.

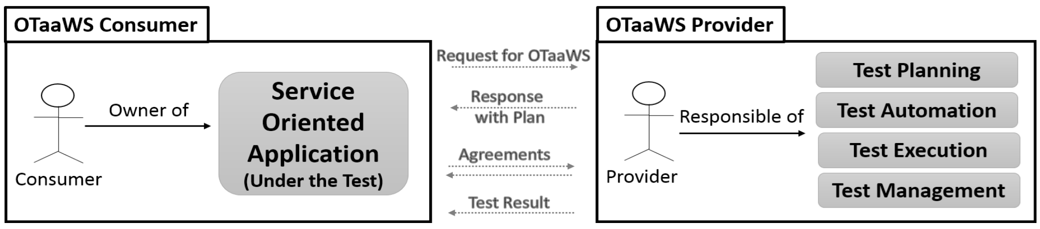


Fig. 1. OTaaWS Contract-base Process (Consumer and Provider side)

In this paper, we tried to combine both test as a service and online test to create a test web service. This web service facilitates the functional testing of service-oriented applications due to changes of its web services. Cost-free web services are publicly available. In this case, the notification of functionality changes of the web service is not sent to the service-oriented application. Therefore, the application may encounter problems when calling its web services.

A. *OTaaWS Phases*

In OTaaWS, the provider side is designed in three phase including Test Automation, Test Execution and Test Management (Fig. 2). The provider will begin its work after the financial and timing agreements with the consumer is done. First, the web services of the application are invoked. Before displaying results of web services to the user, the test web service must be called by the service-oriented application (Consumer), and then the requirements of the service-oriented application, the address of the web service(s), input data and the results (output) of the web service(s) are received from the consumer. With these entries, the content of the test web service, test cases, can be generated. In the Test Execution, if the user selects the online mode, the test web service run the test, at the same time as calling the web services through the application to test their outputs before being displayed to the users. Finally, in the Test Management Phase, test results are sent to the consumer.

1) Test Automation: The main component of the automation phase in the OTaaWS is a Test Case Generation unit (Fig. 3). The test web service contains functional test cases as its content. The top-down generation of test cases can be both in the dynamic and static mode. In the dynamic mode, the content is re-generated every time the service-oriented application calls the test web service. In this case, any kinds of applications with different domain can use the test web service to send their own requirements. On the contrary, in the static mode, the content of the test web service is customized for a specific service-oriented application. The content once generated in the test web service and then it is used by that specific application. As previously mentioned, the test web service receives four entries from the service-oriented application. The web service address and consumer's requirements are the entries to the Test Case Generation unit. In this unit, the type of web service such as researcher is identified through the web service address. Then a section for each web service is created to place related test cases. In order to automatically generate test cases, there must be a mechanism for converting the requirements of the *service-oriented* application into the understandable input for the machine such as grammars and regular expressions. Finally, based on both grammar-based coverage and type of web service, the

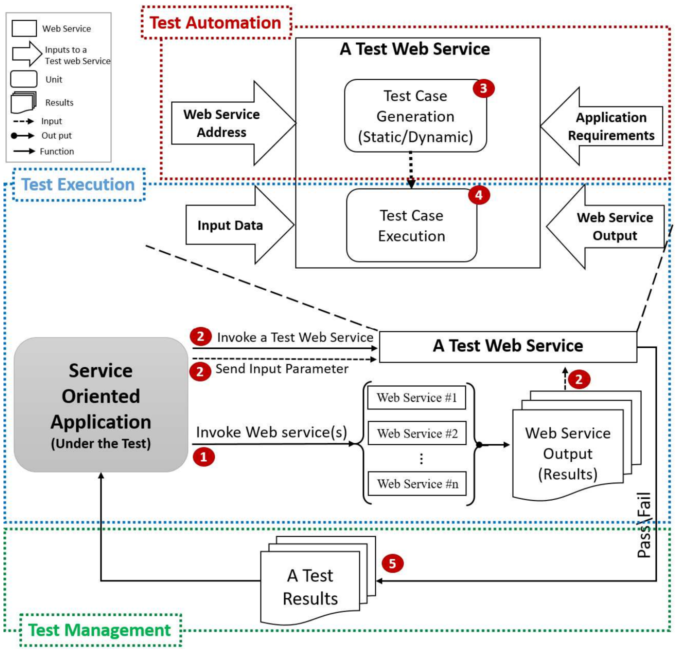


Fig. 2. OTaaWS Process (Provider side)

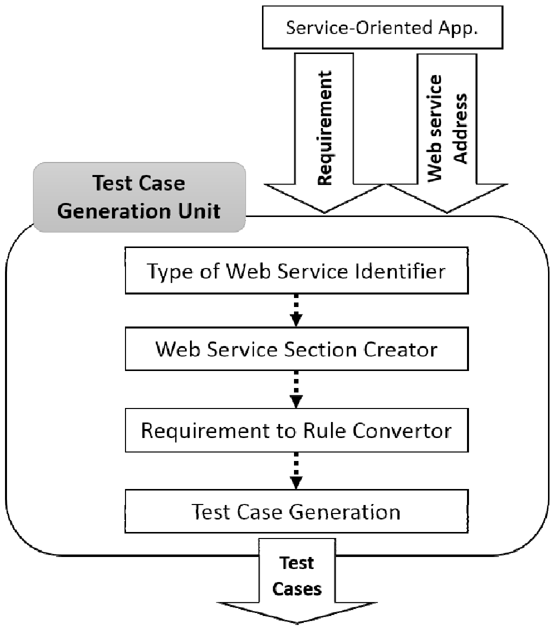

Service-Oriented App.
Requirement
Web service Address
Test Case Generation Unit
Type of Web Service Identifier
Web Service Section Creator
Requirement to Rule Convertor
Test Case Generation
Test Cases

Fig. 3. Test Case Generation Unit

functional test cases for each web service or the integration are generated.

2) Test Execution: This phase of OTaaWS has two separated parts for the execution: 1) web service(s) and the test web service are invoked and executed by the service-oriented application; 2) test cases generated in the earlier phase are executed by a Test Case Execution unit of the test web service (Fig. 4). The input data, the output of the web service and the test cases are entries to this unit. Test cases are run based on the input data and the actual results (output of the web service(s)) are compared with the expected to prepare the test result for the consumer. In this phase, the tests can be executed in an online or offline mode. In other words, the Test Execution can be executed On-demand. By definition, online testing is performed during the operation phase of service-oriented applications. It means that when the application invokes the services to run their activities, the test web service is also invoked. The results of invoking web services, before being displayed in the program, is compared with the expected results provided in the test web service. If there is no defect, the results of web services are displayed in the application, otherwise, based on the outputs of the test web service and the Defect Code Table (Table I), consumer must decide what message should be displayed to the user in the application. In addition, the test web service can be run in the offline mode. In the offline mode, the test web service, can be periodically invoked whenever the consumer requests.

3) Test Management: In this phase, the provider documents the test results and sends it to the consumer. Three different reports including statistics, changes and defects are produced in the Test Management phase. In the. statistical reports, the number of executed test cases totally/per web service, the number of fail/pass test cases totally/per web service and the regression testing are shown In the static generation mode, if the test web service is needed to change due to the changes of service-oriented application and its web services, the changes report is produced. This report is provided for the knowledge management of OTaaWS and will not be sent to the consumer. The defect report is consisted of failed test cases along with descriptions of their defects according to the defect code table.

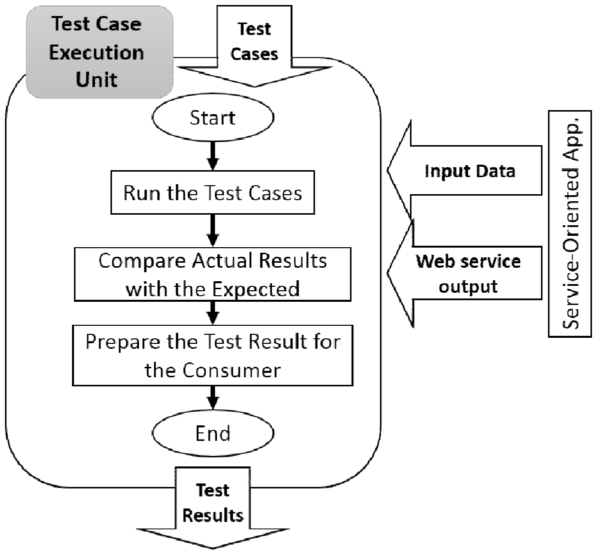


Fig. 4. Test Case Execution Unit

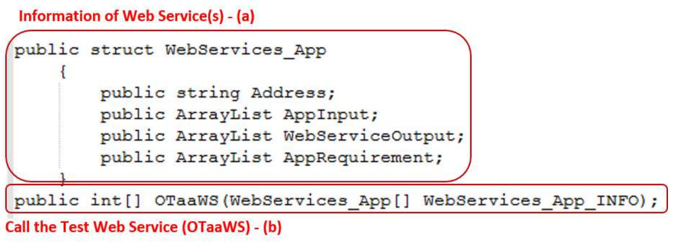
TABLE I. A SAMPLE OF DEFECT CODE TABLE

	Definition	Code
	No Defect	0
Defect related to SOA application	Defect in the Number of Web Service Input	1
	Defect in the Number of Web Service Output	2
	Defect in the type of Web Service Input	3
	Defect in the type of Web Service Output	4
	Defect in the Web Service Address	5
Defect related to web service	Defect in the Web Service Functionality	101 to 199
	Defect in the Web Service Composition (Integration)	201 to 299

B. *Test Web Service Structure*

One of the most important principles in the service-oriented architecture is the reusability of the web service. To this end, the test web service requires a structure for sending and receiving data. This structure is focused on the interactions between service-oriented application and its web services. In Table II, we categorized the mentioned interactions. For example in this category (Table II-Second row), maybe one web service is used in the application. This web service receive two numbers (as the inputs), add them together and send the result number (as the output) to the application. Therefore, the application has one web service with more than one input and one output.

According to Table II, for sending the parameters to the test web service, an array of information for each web service (or a combination of web services) includes a list for the web service address, the input data and the output of the web service, along with the requirements is needed (Fig. 5-a).


Information of Web Service(s) - (a)
Call the Test Web Service (OTaaWS) - (b)

```
public struct WebServices_App
    {
        public string Address;
        public ArrayList AppInput;
        public ArrayList WebServiceOutput;
        public ArrayList AppRequirement;
    }
public int[] OTaaWS(WebServices_App[] WebServices_App_INFO);
```

Fig. 5. Test Web Service Structure

TABLE II. STRUCTURE OF WEB SERVICES USED IN THE APPLICATION

number of web service in an application	number of web service input	number of web service output	example
1	1	1	A web service that finds a number in the list.
	more than one	1	A web service that adds two numbers.
	1	more than one	A web service that finds someone's information based on a code.
	more than one	more than one	A web service that finds someone's profile based on the username and the password.
more than one	1	1	Web services that can be replace with the not working web service
	more than one	1	
	1	more than one	Web services that find different information with a personal code.
	more than one	more than one	Web services that reserve both the hotel and the airplane

The output of the test web service is an integer array that assigns to each web service in the order of input (Fig. 5- b). As we mentioned before, when the test web service is invoked, the test cases are created and executed. If the results of all test cases for each web service are successful, the zero number is entered in the output array. However, if only one test case is failed, a non-zero number, corresponding to the type of defects, is returned as the output of the test result.

V. Case Study And Evaluation

In this paper, a simple web application, namely Get Committee Information is used as a case study (Fig. 6). The Get Committee Information is an application to receive committee information from two different researcher web services, through searching the code of the committee. The first web service provides information on the date of creation and the date of the last modification of the committee's information. The second web service includes general information such as subject, head of committee and activity status.

In this case study, we try the dynamic mode in the Automation phase and the online mode in the Execution phase. By searching the Committee Code, the service-oriented application receives the results of two web services (Fig. 7). Before displaying the results, the service-oriented application calls the test web service and sends its related requirements, addresses of web services, the Committee Code as the input and the outputs of two web services to the OTaaWS (Fig. 8).

After sending the parameters to the test web service, in the Automation phase, the requirements convert to an understandable grammar for the machine to generate test cases (Fig.9). The generated test cases are executed according to the input and output values of the web services and the test results are returned to the service-oriented application.

To evaluate our work, we changed the format of date of creation field (Committee begins) in web service#1. By applying the Test Automation step, 25 test cases for all requirements of two web services are created (Table III). In the Test Execution step, two test cases related to web service#1 are failed due to the differences of the date format in the requirements of service-oriented application with the web service#1.

To evaluate the performance of the OTaaWS process, we pose two research questions. The first research question tries to access the effectiveness of the proposed approach. The second one aims at finding which method has more ability to find the web service failure.

RQ1. What is the effectiveness of OTaaWS?

For evaluating the effectiveness we applied two criteria on the suggested OTaaWS. These two criteria used are as follows:

1- **FDD:** Fault Detection Density is the ratio of the total number of faults detected by test cases (#t_{fi}) and the total number of test cases (#T) [19]. FDD is calculated as (1):

$$FDD = \frac{\sum_{i=1}^{n} \#tf_i}{\#T} \qquad (1)$$

2- **FDE:** Fault Detection Effectiveness is the ratio of the total number of faults (#F_{Total}: A fault that is detected by more than one test case) and total number of unique faults (#F_{Unique}) [20]. FDE is calculated as (2):

$$FDE = \frac{\# F_{unique}}{\# F_{Total}} \qquad (2)$$

The result of FDD is 1 and the result of FDE for the selected case study is 0.08. From the FDD criteria, it can be concluded that each related test case in the OTaaWS (2 test cases for the date creation in web service#1) has ability to find faults. The FDE value implies that all the faults detected are unique and each related test case is focused on one purpose. If this value goes to 1, it indicates the lack of independence of the test cases and their multi-purpose.

RQ2. Which methods have more ability to find the web service failure?

In this part, we compare all the automated methods mentioned in the related work and the OTaaWS with parameters as follows: P1) what is the testing domain? P2) which of the online or offline testing is used? P3) which parts of the testing process is covered? P4) which part of testing process is automated? P5) what is the complexity of the case study? P6) what is the test case generation method?

According to Table IV, our proposed method uses both online and the offline testing, covered all phases of testing process and automated the test case generation and execution compared to other testing methods.

VI. CONCLUSION AND FUTURE WORK

One of the main challenges in SOA testing, as seen in most of the reviewed methods, is the lack of source code for

TABLE III. Number of Test Cases - Case Study

	web service#1	web service#2
Total Test Cases No.	**25**	
Test Cases No.	10	15
Pass Test Cases No.	8	15
Fail Test Cases No.	2	0

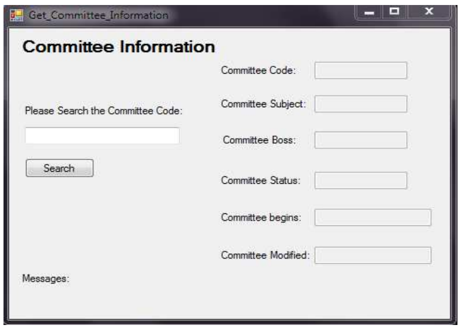


Fig. 6. Get Committee Information Application

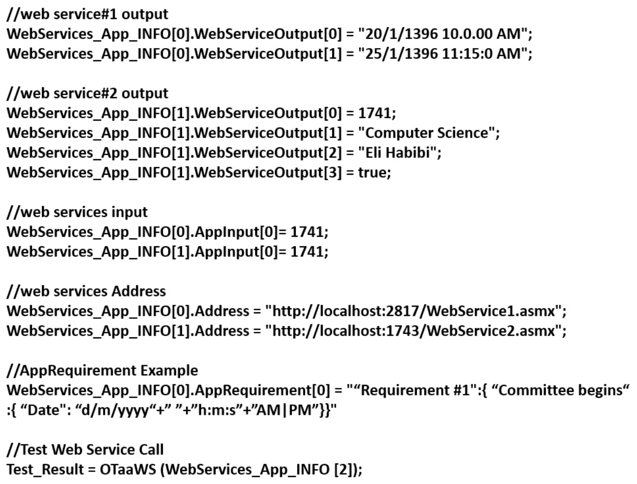
```
ServiceReference1.CommiteeDateInfo C_Date_Result = new ServiceReference1.CommiteeDateInfo();
ServiceReference1.WebService1SoapClient S1 = new ServiceReference1.WebService1SoapClient();
C_Date_Result = S1.Commitee_Date_Information(code);
```

```
ServiceReference2.CommiteeInfo C_Info_Result = new ServiceReference2.CommiteeInfo();
ServiceReference2.WebService2SoapClient S2 = new ServiceReference2.WebService2SoapClient();
C_Info_Result = S2.Commitee_Information(code);
```

Fig. 7. Call Web Services

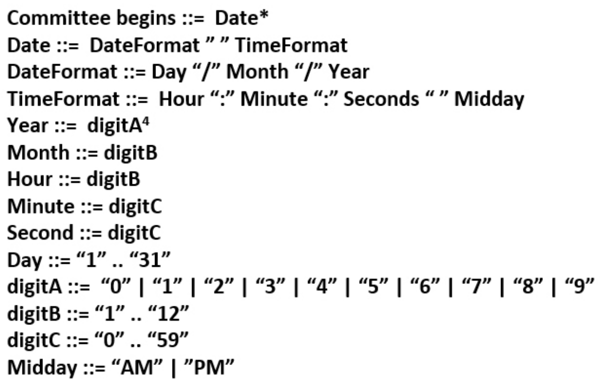
TABLE IV. COMPRAE THE AUTOMATHED METHODS WITH OTAAWS

Parameters / Methods	P1	P2	P3	P4	P5	P6
PROSA	SOA	Online	Test Scenario generation and Test execution		mid	-
Online Platform [11]	-	Online	Test case generation		high	TTCN
metamorphic approach	SOA	Online/ offline	Test case generation		mid	Metamorphic
ConfigTest	SOA	Online	All Phases	Test case execution	low	Event-based
Tools such as SoapUI	SOA	Offline	Test execution	-	-	-
OTaaWS	SOA	Online/ offline	All Phases	Test case generation execution	low	Grammar based



```
//web service#1 output
WebServices_App_INFO[0].WebServiceOutput[0] = "20/1/1396 10.0.00 AM";
WebServices_App_INFO[0].WebServiceOutput[1] = "25/1/1396 11:15:0 AM";

//web service#2 output
WebServices_App_INFO[1].WebServiceOutput[0] = 1741;
WebServices_App_INFO[1].WebServiceOutput[1] = "Computer Science";
WebServices_App_INFO[1].WebServiceOutput[2] = "Eli Habibi";
WebServices_App_INFO[1].WebServiceOutput[3] = true;

//web services input
WebServices_App_INFO[0].AppInput[0]= 1741;
WebServices_App_INFO[1].AppInput[0]= 1741;

//web services Address
WebServices_App_INFO[0].Address = "http://localhost:2817/WebService1.asmx";
WebServices_App_INFO[1].Address = "http://localhost:1743/WebService2.asmx";

//AppRequirement Example
WebServices_App_INFO[0].AppRequirement[0] = "“Requirement #1":{ “Committee begins“
:{ “Date": “d/m/yyyy“+” ”+”h:m:s”+”AM|PM”}}"

//Test Web Service Call
Test_Result = OTaaWS (WebServices_App_INFO [2]);
```

Fig. 8. An example of the test web service entries and call



```
Committee begins ::=  Date*
Date ::=  DateFormat ” ” TimeFormat
DateFormat ::= Day “/” Month “/” Year
TimeFormat ::=  Hour “:” Minute “:” Seconds “ ” Midday
Year ::=  digitA^4
Month ::= digitB
Hour ::= digitB
Minute ::= digitC
Second ::= digitC
Day ::= “1” .. “31”
digitA ::=  “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
digitB ::= “1” .. “12”
digitC ::= “0” .. “59”
Midday ::= “AM” | ”PM”
```

Fig. 9. An example of grammar for the sample requirement

the functional testing. It is also observed that changes in the free web services cannot be easily monitored through the application. Some methods tried to apply online testing on the web services as the solution to this problem. Few methods have been focused on testing as a service. The proposed approach focuses on the composition of both types of testing in the web service environment.

We propose that our suggested testing process has the following advantages: (1) automating test case generation through Test Case Generator; (2) detecting changes in web services through the online test execution; (3) reusing the test cases through the test web service; (4) regression testing through the on-demand test execution; and (5) managing test results to inform consumer. In addition, in our approach, integration test cases are considered due to the complete testing of composing services. OTaaWS is considered as an automatic method, because we created automatic test cases and test web service through the generator.

The suggested testing procedure, like any other methods, has some drawbacks. In the online testing, the response time to the user increases because of the comparison of the results of the services with the expected result in the test web service. In other words, the performance of the application is reduced during the online testing. Therefore, solving this challenge can be one of the future directions of the work for improving our testing process.

ACKNOWLEDGMENT

A Grant from Niroo Research Institute and Sharif University of Technology – Iran, has supported this research.

REFERENCES

[1] W3C Working Group Note, Web Services Architecture, 11 February 2004, Available from: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[2] Endo AT. Model based testing of service oriented applications (Doctoral dissertation, Universidad de São Paulo). 2013.

[3] Wang Y, Stroulia E. Semantic structure matching for assessing web-service similarity. InInternational Conference on Service-Oriented Computing 2003 Dec 15 (pp. 194-207). Springer, Berlin, Heidelberg.

[4] Dong X, Halevy A, Madhavan J, Nemes E, Zhang J. Similarity search for web services. InProceedings of the Thirtieth international conference on Very large data bases-Volume 30 2004 Aug 31 (pp. 372-383). VLDB Endowment.

[5] "Creating bottom-up Web services" and "Creating top-down Web services". Eclipse. Retrieved 2017

[6] Bozkurt M, Harman M, Hassoun Y. Testing and verification in service-oriented architecture: a survey. Software Testing, Verification and Reliability. 2013 Jun 1; 23(4):261-313.

[7] Tsai WT, Wei X, Chen Y, Paul R. A robust testing framework for verifying web services by completeness and consistency analysis. InService-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop 2005 Oct 20 (pp. 151-158). IEEE.

[8] Ammann P, Offutt J. Introduction to software testing. Cambridge University Press; 2016 Dec 13.

[9] Hielscher J, Kazhamiakin R, Metzger A, Pistore M. A framework for proactive self-adaptation of service-based applications based on online testing. Towards a Service-Based Internet. 2008:122-33.

[10] Wang Q, Quan L, Ying F. Online testing of Web-based applications. InCOMPSAC Workshops 2004 Sep 28 (pp. 166-169).

[11] Deussen PH, Din G, Schieferdecker I. A TTCN-3 based online test and validation platform for Internet services. InAutonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on 2003 Apr 9 (pp. 177-184). IEEE.

[12] Chan WK, Cheung SC, Leung KR. A metamorphic testing approach for online testing of service-oriented software applications. International Journal of Web Services Research. 2007 Apr 1;4(2):61.

[13] Bai X, Xu D, Dai G. Dynamic reconfigurable testing of service-oriented architecture. InComputer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International 2007 Jul 24 (Vol. 1, pp. 368-378). IEEE.

[14] Mohanty H, Mohanty JR, Balakrishnan A, editors. Trends in Software Testing. Chapter 7: Testing as a Service, Springer; 2017.

[15] Oracle Enterprise Maanager, Oracle Testing As A Service, Oracle Data Sheet, 2011.

[16] LeanApps Co., Prashant Tapkhirwala, Testing as a service, 2013.

[17] Pardeshi SN, Choure V. Testing as a service on cloud: a review. International Journal on Recent and Innovation Trends in Computing and Communication. 2014 Feb;2(2):188-93.

[18] Bai X, Li M, Huang X, Tsai WT, Gao J. Vee@ Cloud: The virtual test lab on the cloud. InAutomation of Software Test (AST), 2013 8th International Workshop on 2013 May 18 (pp. 15-18). IEEE

[19] R.C. Bryce, S. Sampath, A. M. Memon, "Developing A Single Model And Test Prioritization Strategies For Event-Driven Software", Software Engineering, IEEE, Vol. 37, No 1, pp. 48-64, Utah State University, USA, 2011.

[20] X. Yuan, M. Cohen, A. M. Memon, "Covering Array Sampling Of Input Event Sequences For Automated GUI Testing", 22nd International Conference On Automated Software Engineering (ASE), ACM/IEEE, pp. 405-408, Nebraska-Lincoln and Maryland University, USA, 2007.