I.J. Information Engineering and Electronic Business, 2012, 5, 1-8
Published Online October 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2012.05.01

Test Cases Reduction and Selection Optimization in Testing Web Services

Izzat Alsmadi
Computer Information Systems Department, Yarmouk University, Irbid, Jordan
ialsmadi@yu.edu.jo

Sascha Alda
Department of computer science, Bonn-Rhein-Sieg University, Sankt Augustine, Germany
Sascha.alda@h-brs.de

Abstract — Software testing in web services environment faces different challenges in comparison with testing in traditional software environments. Regression testing activities are triggered based on software changes or evolutions. In web services, evolution is not a choice for service clients. They have always to use the current updated version of the software. In addition test execution or invocation is expensive in web services and hence providing algorithms to optimize test case generation and execution is vital. In this environment, we proposed several approach for test cases' selection in web services' regression testing. Testing in this new environment should evolve to be included part of the service contract. Service providers should provide data or usage sessions that can help service clients reduce testing expenses through optimizing the selected and executed test cases.

Index Terms — SOA, web services, regression testing, test case reduction, software testing, Automation

I. INTRODUCTION

Many software providers consider providing their software as a web service. There are many factors that promote going into this direction. For service providers, cracking and software piracy is minimal in comparison with traditional software acquisition methods. The availability of the Internet and the fast speed and large bandwidth also helped in the feasibility of such option. For customers, they can get continuous support and evolution. This is since customers always access and use the current version of the provided software. Using rather than owning, is what distinguish web services from traditional software applications.

Software testing is a major software development stage aims at making sure that developed software products have the expected features by their users and with minimal number of errors. Testing a software product includes several activities that may occur in several stages in the software development process. In fact, every deliverable in every stage should be tested. For example, before proceeding from the requirement to the design stage, requirements analysis, gathering and specification should be tested and reviewed with customers to make sure that no misunderstandings or misinterpretations occurred in the collection and analysis process. Testing in this stage should verify that the will be developed software is feasible and achievable given the company existed resources, etc.

Regression is one of the testing activities that usually occur late in the software development. Test cases are created to set software aspects and components whether black box to test high level functionalities, or white box to test low level code. Those test cases form a test set or test database. This database is continuously evolving and should be updated when new code or features are added to the software product. This testing database, also called test oracle, form a baseline that goes in parallel with the code to make sure that any new changes or additions to the software product may not break earlier or existed components. In this scope, regression testing is running the testing database or oracle frequently or when changes or updates occur in the software product.

In testing web services, regression testing has some new aspects. This is since evolution in web services can be continuous and dynamic. Users or customers don't get officially frequent releases. Updates on the web service occur on the server side and users will see them in the next time usage or request of this service. Perhaps, there are some constraints and regulations on such continuous evolution and release form especially if such change will change the Web Service Description Language (WSDL) or one of its components. WSDL XML file is the main mediator or facilitator between service provider and consumers. The file or document contains all public web methods in the subject web service. For each one of those public methods, the WSDL also includes information related to the inputs or parameters for those methods and their expected output. New versions of WSDL such as WSDL Semantic (WSDL-S) try also to add more semantic information such as the pre- and post-conditions for each one of the available methods.

The WSDL then represents the web service interface that others can see and call the service through.

In the scope of web services also, regression can be divided physically into two parts: one that should be conducted on the server side and another or others that should be conducted on the client side or sides. This is since one standard service can have different ways of usage from the client side and hence different client can have different regression databases.

Evolution in services is eminent. They are developed to be continuously adaptable and used by a large spectrum of users. Testing in web services should also evolve from the concepts of complete comprehensive testing to continuous, incremental and proactive testing.

In real business scenarios, service composition is used to combine several possible services to accomplish a particular business task. This forms another evolution dimension where changes may not occur to a service or services, but the composition structure is changed, one or more services entered or left the composition. This may encourage the need to run regression database all over again.

In software products, errors can occur for different purposes. They may indicate software internal problems. They may also indicate misunderstanding of the software or the service especially when users enter incorrect or invalid inputs. Errors in particular or quality problems in general can be also do to network problems or traffic congestions, especially in web services and distributed systems. In web services, Service Level Agreement (SLA) is a contract with quality related specifications that identify to service users what to expect on their side when using or invoking a service.

In testing web services environment, when a user wants to test a web service, he/she needs to invoke the service part of testing activities. In most cases, users are charged for services per invocation, this can be problematic to testing and may risk lowering the number of test cases at the cost of quality.

Test case reduction and selection techniques may then be more urgent in web services' testing environment in comparison to traditional testing environments. It is important then for the client to be able to test the service with the least and best number of selected test cases that can increase coverage and find possible weaknesses or errors in the web service.

Coverage or adequacy is a term used in testing to define the percentage that is covered by a testing suite. This percentage is applied to the software product or one of its components: code, requirements, user interface, etc. To compare a good test cases' selection from another, coverage is an important factor where if the number of test cases in two test sets is the same, the one that has a better coverage is selected.

There are many coverage aspects in software products. A test set may focus on covering code statements, branches, paths. It may also try to cover all possible inputs from the user interface. For web services, the Web Service Description Language (WSDL) file that is published about the service can be the first coverage focus from the client side. Testing from the client side can either start from their client user interface or application, or it can start from testing the web service through its WSDL.

WSDL contains all public methods in the web service. For each publish method, WSDL contains further the method input parameters and expected or possible output. Some WSDL extensions such as: WSDL-semantic may contain semantic information related to the usage or calling of those methods such as: pre- and post-conditions. Changes that service provider may apply to the software can be only seen by service consumers if WSDL is changed. If changes cause no change for any part of the interface, such evolution is considered internal and may not need testing database update. However, regression testing should be periodically executed to make sure such changes did not cause any problem.

As coverage starting point for testing based on WSDL, ever input for every method should have at least one test case. For example, if a web service S, have several methods (m1, m2, … mn), and each method m, has several input parameters (i1, i2, … ik), then, the minimum number of test cases (TC) required for coverage can be calculated through the equation:

$$TC = \sum_{n=1}^{m} \sum_{i=1}^{K} (\text{input parameters})$$

We did not mention method outputs because they will be included in the number of inputs. Reduction can also occur since inputs parameters occur in combination and hence minimum coverage can go down to the number of methods only.

Of course such minimum may not guarantee exclusion from all possible errors. Some traditional methods for input parameters divide input parameters into valid and invalid domains and require creating test cases for both. This will double the number of minimum test cases for coverage in the previous equation.

In some cases, a particular user may not be interested on all service methods. Their focus is on one method to test it extensively before using it and make sure it is reliable to use.

Test Driven Development (TDD) is a recent concept or approach that appears especially in agile software development models. Opposite to the norm where the testing stage comes after the coding or development stage, in TDD, test cases are written first and then code is developed to fulfill those test cases (no more, no less). In web services, this concept can be used in a different scope. Since service consumers are distant from service providers and since services are developed to be generic and not based on a specific client scope, test cases can be created and provided

from the service provider as part of the service contract. Those test cases can be the baseline for the contract where service providers showed their constraints and guidelines, through those test cases, for using their service.

Coverage assessment can be accomplished statistically or dynamically. In statistical approaches for coverage assessment, a software product aspect or component is selected and test set coverage is then calculated based on that aspect. For example, for path coverage, all software possible paths are calculated. Path coverage for a particular test set is then the result of dividing the number of paths visited in the test set to the total number of paths in the software product.

Dynamic coverage assessment requires first executing test cases and then calculates coverage based on execution results and faults detection. While dynamic coverage is more realistic in representing the actual software quality state, it is also considered harder and more expensive to accomplish.

The rest of the paper is organized as the following: In section 2, several samples of related work to the paper subject are presented, section three described methodology, experiments conducted along with their analysis. The paper is concluded with a conclusion and possible future work.

II. RELATED WORK

While testing in web services is relatively a new research subject, several research papers discussed different aspects in this area. In this section, we will focus on those papers discussed test cases reduction and regression testing in web services in particular.

While most research papers in this area tried to utilize regression testing and test case selection algorithms in traditional methods, however, there are many factors that make those activities in web services' testing different. First, users have no access to source code and source code is physically distant from users. Second, dynamic users' interactions in real time web services' usage may arise some problems or issues that cannot be tested statically from one client. Further, service composition issues are unique and error localization can be problematic in such cases.

(Tarhini et al 2006, 2008 [1, 2]) papers discussed a theoretical approach for selection of test cases for web services and applications regression testing. The system is modeled as a timed label transition system or state machine and tried to select unique test cases that can best represent the regression database. Authors discussed different possible modifications that may occur to a web service, a composition of web services, or a web application: adding new components, modifying an existing component, or changing the specification for the web service component.

In Penta et al 2007 [3] book chapter, and Bruno et al 2005 [6] paper, authors discussed issues and challenges related to regression testing in web services. They also proposed a test regression strategy for testing evolved web services based on Quality of Service (Qos) or Service Level Agreement (SLA) constraints. As service users has no direct access to the service and have limited options for test executions, authors suggested that service providers should publish in addition to the service and interface, samples of valid test cases as well as usage sessions or logs. Test cases as a contract, a concept that was discussed in many research papers that discuss testing in web services environment specially to deal with the problem that test execution can be very expensive.

Optimization research, integer and linear programming were used in some papers for optimizing the selection of test cases in web services (Hou et al. 2008 [4]). In web services, users are charged per usage and they are usually given a usage quota per month or unit of time. Authors formulated a cost-constraint problem with usage quota as a goal for such optimization problem. Matrix attributes include: test suite, service composition group, testing requirements or constraints, request quotas, test requirement coverage and web service invocation matrices. In reality, formulating and finding all those values for a particular case is a challenging and complex task. Even the time slot calculation is also not simple especially as such time is usually for usage and only portion of it can be dedicated to testing.

Ruth et al 2006 [5] paper proposed a white box approach for safe regression test cases' selection applied on Java web services. A control flow graph is developed from Java source code. Traceability analysis is conducted between current and earlier versions of the service to see the areas of the code that we affected by the evolution. Such scenarios can be tested on experimental web services. In reality, clients have only black box testing options when it comes to web services as they can have no access to the code.

Blanco et al 2010 [7] discussed an approach (scatter search technique) for the automatic test case generation from web services' business processes. The approach depends on search for unique transitions in the business processes to optimize test cases' selection. A state graph is modeled for the business process and coverage is then extracted based on visiting all transitions in the state graph. While WSDL XML files represents the service side, business processes (BPEL) represents the client side. To deal with special state cases, such as loops and exceptions, authors proposed a transformation model to represent those special cases.

Athira and Samuel paper 2010 [9] discussed a model based test case prioritization in testing web services. This traceability method is based on studying portions of the service that were affected by evolutionary changes. Testers need to archive testing logs for earlier versions to be able to trace execution logs and see what is the different between current and previous execution logs. The approach is discussed theoretically based on tracing events in an activity diagram.

Zhai et al 2010 [10] paper discussed regression testing and test case selection particularly in dynamic service composition. Service selection is included as an

important factor in this process where executed test cases will be reduced through binding a particular client only to the candidate service provider. Authors also some location based metrics to point to the areas of possible testing focus. The testing and metrics approach is applied on a case study of "city guide" web service composition to provide users based on their locations with points of interests such as hotels, restaurants, etc.

Authors further elaborated some unique aspects in web services evolution where users have no control on such evolution or when it can or should occur. The next time users invoke the service, they will get the last version. Regression testing activities in traditional software development environments are called whenever the software goes in a cycle of change, update, etc. In web services, this can be set to periodic especially from the client side, since client is not aware of such evolution cycles.

Authors of Mei et al 2009, 2011, 2012 [11, 12, and 13] papers have also their contribution in the area of web services regression testing. In those papers, authors discussed issues related to when to trigger regression testing and how to make sure that selected test cases are up to date and are best representatives for the testing database. In the service composition scope, one component service may evolve while the rest of the services may not be aware of this evolution. To avoid possible synchronization problems, algorithms are proposed in those papers to dynamically inform service composition team of component service changes.

Nguyen et al 2011 [14] paper discussed using Information Retrieval (IR) techniques for web services test case prioritization. The paper focused on audit testing for evolution in web services' composition. Test cases are prioritized based on their relevancy to the service change behavior. Given a service change or new features as known, a search is conducted through the test cases to find the best test cases that can assess the new changes. However, in reality, the success of such approach may depend on the format that the changes and the test cases are in, their level of abstraction and how much relevancy can be drawn between test cases and evolution document.

III. GOALS AND APPROACHES

The experiments in this paper tried to answer the following questions:

I. How can we reduce the number of executed test cases without necessary reducing the number of generated test cases?

II. How can application data be verified, when the application is a Web Service without a Graphical User Interface (GUI)?

III. Is there any difference in terms of load testing between many calls from many users and many calls from single user? Do virtual users mean one after another or many threads parallel?

- How can we reduce the number of executed test cases without necessary reducing the number of generated test cases?

3.1. Preliminary validation components

In web services environment, test execution or invocation is expensive, as in most cases, service revenue models are based on number of invocations. In principle, no problem or limit to the number of test cases that are generated.

On the other hand, most coverage calculations depend on calculating the number of faults found by the test cases and not the number of test cases. The number of faults, or any related attribute are in the numerator, while the number of test cases are in the denominator. This means that to optimize coverage we have to increase the number of faults or possible faults found and/or decrease the number of generated or executed test cases.

A possible solution to trade off between the need to increase coverage and at the same time reduce execution cycles is in creating a pre-execution component on the client side to perform initial validations on the generated test cases before possible invocation or before validating them for execution. The goal of this mediating component is to make sure that allowed generated test cases to be executed are not going to be redundant and will improve one or more coverage aspects, specially fault detection.

Those components will have rules or constraints to validate test data and method selection of input parameters' values. Figure1 depicts the high level design of a regression testing model for testing web services based on the pre-validation component. The pre-validation model takes the log of previously executed test cases as input to help in defining roles for judging valid and invalid test cases. Figure1 shows a possible structure showing the location and role of the testing pre-validation component.

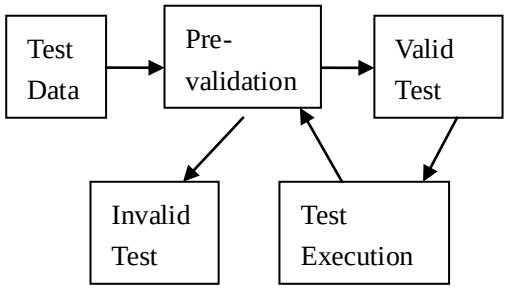

Figure 1. Pre-validation for test cases' selection.

This pre-validation process can be automated. The loop back from the test execution results to pre-validation ensure tuning the constraints on test cases based on results from previous test cases. In reality, if logs are available from user sessions, those can be also used to optimize the usage of test execution and minimize the number of invocations. Log reports from web service invocations include information related to the called method, parameters, and output. The initial set of test cases can be provided by the service provider as part of the contract or as a user manual for how to call and use their service.

Figure 2. A sample of web service execution log or reports

We developed a test automation tool to automatically execute bulks of test cases and report the results. Figure 1 shows an except output from a test session for web service invocations. To clarify the results, we will show to reports invocations, one for a successful invocation and another for a filed invocation. Figure 2 shows the output report for a successful invocation from invoking the service (http://www.webservicex.net/uszip.asmx?WSDL), the method: GetInfoByZIP. This method takes the zip code as input parameter and retrieves the city, state information were this zip code is in. Figures 3 and 4 show two test cases with valid and invalid inputs.

```
http://www.webservicex.net/uszip.asmx?WSDL,USZip,
GetInfoByZIP,06401,"<CITY>Ansonia<\/CITY>
<STATE>CT<\/STATE><ZIP>06401<\/ZIP>
<AREA_CODE>203<\/AREA_CODE>
<TIME_ZONE>E<\/TIME_ZONE>
<\/Table><\/NewDataSet>"
```

Figure 3. A test case with valid input

```
http://www.webservicex.net/uszip.asmx?WSDL,USZip,
GetInfoByZIP,CC,error,Object reference
not set to an instance of an object.,
```

Figure 4. A test case with invalid input.

An automatic validation tool that parses through execution log reports can easily distinguish between a test case that passes and the one that fails. Based on Figure1 then, report feedback can be used as an input to the pre-validation component to control or decide the test cases that should be validated to enter the execution process.

1.2. Offline or archive testing

Testing in traditional software development environment is an expensive stage. It consumes a significant amount of project time and resources. Further, in web services, testing from the client side, means calling or invoking the web service. This is usually a charged service where most revenue models for web service charge per service invocations. Such issue can be problematic for both service providers and consumers. For service providers, those are not real usage and required to make sure service is functioning as expected and hence paying for testing can be unjustifiable. Further, they may reduce testing activities and risk quality problems. For service consumers, even if this is charged, this may interfere with normal usage, specially as many testing activities may require abnormal scenarios (e.g. stress testing, fault based testing) which may risk crashing the web service.

To solve such problem, we propose off-line testing approaches. First for most testing activities, real data is not a must. Historical data from web service logs can be very useful for many activities such as test case generation, prioritization, regressing testing, results verification, etc. A significant portion of testing activities can be conducted based on historical data. In our previous regression testing model, historical data can be used to generate test data and hence optimize the process of test case generation.

Initial test cases and test requirements or constraints can be provided by the service provider to their client as part of the service contract. Their monitoring systems or logs can have very valuable information for testing. The only possible problem with distributing such log is privacy especially if such data include clients' private information.

Further, historical data in logs can be very useful also for regression testing. Log analysis can help in tracing possible problems due to service evolution.

1.3. How can application data be verified, when the application is a Web Service without a Graphical User Interface (GUI)?

In most test automation frameworks for testing web services, verification stage for black box testing is accomplished manually. There are some unit assertion tools that can automate test results verification based on comparing expected results with actual ones that came as outputs from execution reports.

However, in black box functional testing, automatic verification is a challenging process. First, this is since it is complicated in many scenarios to describe expected results. Second, since some actions need complex user interactions that can hardly be simulated or automated.

There are direct and indirect goals for test cases' verification. The final indirect goal is to make a simple judgment whether the test case passes or fails. The direct goal is to measure the results of a test case execution and compare it with expected results (after defining expected results in an earlier stage) to come up with the Boolean judgment of whether the test case passes or fails.

For coverage perspectives and to optimize the process we propose a process to jump to the indirect process and judge whether the test case passes or fails without looking at evaluating complex results. In this way, two complicated activities will be skipped: First defining expected results for the test cases, and second making the comparison through the extensive analysis of execution reports in comparison with initial assertions.

For testing web services, we noticed that in many testing areas, especially in fault-based approaches, execution reports from faulty test cases, produce

exception output that can easily be distinguished from successful reports by a test automation tool.

1.4. Is there any difference in terms of load testing between many calls from many users and many calls from single user?

Many testing tools specially for testing web services allow users to simulate load testing in their ability to emulate many virtual threads or users. Tools claim that those users run in parallel. Each thread can have a separate HTTP connection. But whether those virtual threads run independent from each other or not will depend on the service provider and the setup or the platform.

In regression testing however, we need to have one user with an ability to send many requests (parallel or serial). A test set that has for example 100 test cases to execute, will need to send those as service requests one after another or all in parallel. Those two options (i.e. one after another, or all at once) can produce different results not only in terms of speed or performance, but sometimes in terms of output results especially if service provider has a mechanism to control robotic requests, simultaneous users, etc.

Available load testing tools for web services such as Jmeter may not provide the flexibility to send a bulk of test cases, serial or parallel with complex customization for each test case to reduce redundancy. As such, we developed a tool to automatically execute a test set that contains many test cases and record the invocation reports.

One issue regression testing activities may face is that some web services may disturb a tool that is trying to send a bulk of service invocation, serial or parallel. We noticed that some of the evaluated web services block response on such bulk service invocations.

Figure 5 shows invocation report generated by our developed tool from calling a service. Those test cases are invoked one after another. With the exception of the first one, due to possibly communication overhead, the total response time for all next service invocations is somewhat similar.

WSDL	Service	Method	Input	Results	Time(ms)
http://ww	USZip	GetInfoBy	2101	<NewData	1992
http://ww	USZip	GetInfoBy	2102	<NewData	752
http://ww	USZip	GetInfoBy	2103	<NewData	727
http://ww	USZip	GetInfoBy	2104	<NewData	740
http://ww	USZip	GetInfoBy	2105	<NewData	729
http://ww	USZip	GetInfoBy	2106	<NewData	755
http://ww	USZip	GetInfoBy	2107	<NewData	805
http://ww	USZip	GetInfoBy	2108	<NewData	751
http://ww	USZip	GetInfoBy	2109	<NewData	740
http://ww	USZip	GetInfoBy	2110	<NewData	740
http://ww	USZip	GetInfoBy	2111	<NewData	737
http://ww	USZip	GetInfoBy	2112	<NewData	747
http://ww	USZip	GetInfoBy	2113	<NewData	756
http://ww	USZip	GetInfoBy	2114	<NewData	839
http://ww	USZip	GetInfoBy	2115	<NewData	748
http://ww	USZip	GetInfoBy	2116	<NewData	744
http://ww	USZip	GetInfoBy	2117	<NewData	748
http://ww	USZip	GetInfoBy	2118	<NewData	739
http://ww	USZip	GetInfoBy	2119	<NewData	765
http://ww	USZip	GetInfoBy	2120	<NewData	732
http://ww	USZip	GetInfoBy	2121	<NewData	744
http://ww	USZip	GetInfoBy	2122	<NewData	779

Figure 5. Invocation report for testing a web service

1.5. Exception Handling and management

In software testing, we are not only concerned in testing with valid inputs only, fault based approaches try to invalid method input parameters and evaluate service exception handling and management. Figure6 shows results from our tool test invocation results using invalid input for different tested services. It is noticed that many web services are still immature in this area and their exception messages or feedback is in many cases uninformative and do not reflect the real invalid case to help service users understand what went wrong.

```
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl,ndfdXML,
LatLonListZipCode,02101,error,Parameter count mismatch.,,0
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl,ndfdXML,
LatLonListZipCode,2101,error,Exception has been thrown by the target of an invocation.,,0
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl,ndfdXML,
LatLonListZipCode,02101,"<?xml version='1.0'?><latLonList>42.3583,-71.0603<\/latLonList><\/dw
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl,ndfdXML,
LatLonListZipCode,,,error,Exception has been thrown by the target of an invocation.,
http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl,ndfdXML,
LatLonListZipCode,103,error,Exception has been thrown by the target of an invocation.,
http://www.webservicex.net/weatherForecast.asmx?WSDL,WeatherForecast,
GetWeatherByZipCode,02101,{"allocationFactorField":0,"detailsField":null,"fipsCodeField":null
"longitudeField":0,"placeNameField":null,"stateCodeField":null,"statusField":null}
http://www.webservicex.net/uszip.asmx?WSDL,USZip,GetInfoByZIP,02101,"
<NewDataSet><Table><CITY>Boston<\/CITY><STATE>MA<\/STATE><ZIP>02101<\/ZIP>
<AREA_CODE>617<\/AREA_CODE><TIME_ZONE>E<\/TIME_ZONE><\/Table><\/NewDataSet>"
http://www.webservicex.net/uszip.asmx?WSDL,USZip,GetInfoByZIP,,
error,Object reference not set to an instance of an object.,,
http://www.webservicex.net/uszip.asmx?WSDL,USZip,GetInfoByZIP,,
```

Figure 6. Exception response from different services' invocations

1.6. Service reuse or extension

Service consumers can be end users or mediators. In some cases, composite services can be built on atomic or component services. Extending a service is like extending a class in traditional software programming. There are many roles that were proposed to control extensibility and inheritance. For example, design patterns such as: Liskov Substitution Principle (LSP), Design By Contract (DBC) and Open Close Principle

(OCP) all aim to control or put constraints on how to extend from a parent class. In service reuse or extension principles and focusing on pre- and post-conditions for service public methods, the child service should demand no more and promise no less in reference with original services. Such roles can be applied on input parameters or test data in test case generation process.

II. RESULTS AND CONCLUSION

Software testing is a high level generic activity that is required for any type of software regardless of its nature, development methodology, target, etc. We evaluated in this paper regression testing activities and their challenges in testing web services. Test cases reduction is more urgent in testing web services in comparison with such reduction in testing traditional software applications. Focus should be on optimizing test executions only as all other activities can have more flexibility in terms of resources' availability. Two proposals we focused on in this paper; the first one is to develop a pre-test execution component that can evaluate generated test cases and optimize the selection from those generated test cases for execution. We also proposed the utilization of historical usage sessions that can be provided to clients by service provider. Such usage sessions can direct and optimize the process of test cases' generation and execution.

ACKNOWLEDGEMENT

This paper is conducted part of postdoc scholarship for AVEMPACE (Erasmus Mundus, Action 2 Strand 2 Lot 5 AVEMPACE) in 2012.

REFERENCES

[1] Tarhini, Abbas, Hacene Fouchal, and Nashat Mansour, Regression Testing Web Services-based Applications, AICCSA '06 Proceedings of the IEEE International Conference on Computer Systems and Applications, Pages: 163-170, 2006.

[2] Tarhini, Abbas, Zahi Ismail, and Nashat Mansour, Regression Testing Web Applications, 2008 International Conference on Advanced Computer Theory and Engineering, 2008.

[3] Massimiliano Di Penta, Marcello Bruno, Gianpiero Esposito, Valentina Mazza, Gerardo Canfora: Web Services Regression Testing. Test and Analysis of Web Services: Pages 205-234, Springer, 2007.

[4] Shan-Shan Hou, Lu Zhang1; Tao Xie, Jia-Su Sun, Quota-Constrained Test-Case Prioritization for Regression Testing of Service-Centric Systems, In Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2008).

[5] Ruth, Michael, Feng Lin, and Shengru Tu, Applying Safe Regression Test Selection Techniques to Java Web Services, International Journal of Web Services Practices, Vol.2, No.1-2 (2006), pp. 1-10.

[6] Marcello Bruno, Gerardo Canfora, Massimiliano Di Penta, Regression Testing of Web Services, In CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, 2005.

[7] Raquel Blanco, Jos é Garc á-Fanjul, Javier Tuya, Test case generation for transition-pair coverage using Scatter Search, International Journal of Software Engineering and Its Applications Vol. 4, No. 4, October 2010.

[8] S. Yoo, M. Harman, Regression Testing Minimisation, Selection and Prioritisation : A Survey, SOFTWARE TESTING, VERIFICATION AND RELIABILITY Softw. Test. Verif. Reliab. 2007; 00:1–7 (DOI: 10.1002/000).

[9] Athira, B, and Philip Samuel, Web Services Regression Test Case Prioritization, 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), 8-10 Oct. 2010.

[10] Ke Zhai, Bo Jiang, W. K. Chan, T. H. Tse, Taking Advantage of Service Selection: A Study on the Testing of Location-Based Web Services through Test Case Prioritization, ICWS 2010.

[11] Lijun Mei, Ke Zhai, Bo Jiang, W. K. Chan, T.H. Tse, Preemptive Regression Test Scheduling Strategies: A New Testing Approach to Thriving on the Volatile Service EnvironmentS, Proceedings of the 36th Annual International Computer Software and Applications Conference (COMPSAC 2012), 'Trustworthy Software Systems for the Digital Society', Izmir, Turkey, 16-20 July 2012.

[12] Lijun Mei, Zhenyu Zhang, W. K. Chan, T. H. Tse: Test case prioritization for regression testing of service-oriented business applications. WWW 2009: 901-910.

[13] Lijun Mei, W.K. Chan, T.H. Tse, and Robert G. Merkel, XML-manipulating test case prioritization for XML-manipulating services, Journal of Systems and Software 84 (4): 603–619 (2011).

[14] Cu D. Nguyen, Alessandro Marchetto, Paolo Tonella, Test Case Prioritization for Audit Testing of Evolving Web Services using Information Retrieval Techniques, ICWS 2011: 636-643.

[15] Lin Chen Ziyuan Wang Lei Xu Hongmin Lu Baowen Xu, Test Case Prioritization for Web Service Regression Testing, 2010 Fifth IEEE International Symposium on Service Oriented System Engineering.

Izzat Alsmadi is an associate professor in the department of computer information systems at Yarmouk University in Jordan. He obtained his Ph.D degree in software engineering from NDSU (USA) and his second master in software engineering from NDSU (USA) and his first master in CIS from University of Phoenix (USA). He had a B.sc degree in telecommunication engineering from Mutah University in Jordan. He has several published books, journals and conference articles largely in software engineering and information retrieval fields.

Sascha Alda is a professor in software architecture at the University of Bonn-Rhein-Sieg (Sankt Augustin). He got a Dipl. Information degree from University of Koblen-Landau and Leiden University in 2000. In software architecture, his current research topics include: end-user customization, model based management, and several related subjects in SOA. Other research topics include: architectural patterns for mobile devices, and workflow architectures for different applications.

