A Survey on Testing SOA Built using Web Services

Poonkavithai Kalamegam and Zayaraz Godandapani

Department of CSE, Pondicherry Engineering College, Pondicherry, India
poonks2012@gmail.com, gzayaraz@pec.edu

Abstract

Service Oriented Architecture (SOA) is generally defined as a business-centric IT architectural approach that supports integrating businesses as linked, repeatable business tasks, or services. The successful deployment of SOA implementation in any enterprise is ensured by testing of applications. An inspection of recent research achievements related to SOA testing is present in this paper. Testing challenges from the viewpoint of different stakeholders along with different levels of testing, including unit, integration, end to end and regression testing are analyzed. Further the ways to improve functional testing of SOA applications created using Web Services is explored.

Keywords: SOA, SOA Testing, Web Services

1. Introduction

SOA is a design pattern which is composed of loosely coupled, discoverable, reusable, inter-operable platform agnostic services in which each of these services follow a well defined standard [8]. Each of these services can be described, published, categorized, discovered and dynamically assembled, be bound or unbound at any time and as needed. As a result, testing and performance change greatly when a corporation moves to SOA. Enterprises who are in the early stages of their SOA initiatives or those contemplating starting a SOA initiative should also address the impacts of SOA as it pertains to testing. Success of any deployment in an enterprise depends on the quality assurance process undertaken. SOA testing is quite different from traditional testing. It requires its own type of test architecture and tester skills. Even the best of firms have taken several years to figure out the best way to test SOA implementation. To test any service-oriented architecture, all the components needs to be tested in isolation along with the common use cases where the systems are interdependent. SOA testing spans several levels taking into account both functional and non-functional aspects of the individual services and the inter-enterprise federations of systems. A comprehensive and maintainable testing methodology is critical for the successful functional testing of a service-oriented system.

Devising testing strategies for service-oriented system is still a young research area. SOA testing strategy must encompass infrastructure components, individual and composite services and end-to-end business processes, spanning multiple services and applications. The infrastructure components are the capabilities that aid in registering and discovering services, providing security and delivering messages. The SOA infrastructure may be constructed using custom developed software, commercial products, or a combination of both. Web services might be atomic or composition of services utilised to represent reusable business functionality. A web service testing involves testing of the service interface, implementation of service, message exchange and service level agreement (SLA). Composite services are built by assembling multiple web services that interact to meet a business requirement. End-to-end business sequences must cover all the applications, services, back-end systems and
databases that are essential to perform a business process. Moreover it should also take into account the perspective of end users and business clients, infrastructural aspects and the operating environment. The main purpose of the different levels of testing namely unit, integration, system, and regression testing is to assure the system will deliver the expected functionality. However, the dynamic and adaptive nature of SOA makes most of the existing traditional testing techniques not applicable to test services and service-oriented systems. In this paper the existing testing techniques that have been proposed by several people has been presented. The remainder of the paper is organized as follows. Section 2 explains challenges in SOA Testing. Section 3 briefs about the elements to be tested and Section 4 explains role of different stakeholders involved in testing activities. We present the several existing techniques at different levels of functional testing in Section 5, followed by recommendations for testing in Section 6. Finally Section 7 concludes the survey.

2. Challenges in SOA Testing

An SOA implementation would be cost-effective only if it is properly validated. Robustness is the key attribute of all SOA implementations that has be achieved; otherwise, the power of SOA is not fully realized. Testing service-oriented applications or SOA implementations is challenging by a number of factors because such systems are componentized, distributed and moreover are provided by different organizations. Mostly the services and their providers are often outside the control of the consumer leading to potential mismatches and misunderstandings. The unanticipated evolution of services, varying load on services, flexible infrastructure and the underlying network makes SOA implementations highly dynamic. This makes replication of all possible configurations and loads during the testing process very costly and risky. The testing of such large distributed system is a significant task and an appropriate test plan is needed to perform the tests smoothly and efficiently. Investigating the atypical characteristics of SOA that pose challenge to testers and the testing process will enable design of effective and efficient test strategy. Listed below are the primary challenges to testing a service-centric solution [1, 2].

2.1 Need for Strong Technical Knowledge

SOA requires a combination of black, white, and gray box testing since each layer is bounded by many factors of the architecture. For testers who have been practising more traditional white box and black box testing SOA implementations would be very new. If not the technicalities the need and logic of how to work with each layer; Application Service Layer, Business Service Layer and Orchestration Service Layer must be known. This demands ample amount of scripting and/or coding knowledge along with understanding of SOA concepts. Mostly testers who are experienced in the technical aspects might overlook the business need for the SOA implementation and vice versa. A SOA test engineer needs to understand all interfaces; the business need for service orientation and must have strong technical knowledge.

2.2 Lack Visibility into the Overall Strategy

Thus far there is no guidelines on “how and where to start” for tackling SOA. SOA is still young and in fact, many people do not get the correct idea about what SOA is. Because of these issues, many training materials and tools are not up to par. Testers struggle with the concepts of the layers in SOA and to identify appropriate testing approaches. Many testers tend to apply the testing practices that they are comfortable with which might be
inappropriate for testing service-oriented systems. Moreover the root-cause analysis of the defects unearthed in the SOA space is extremely difficult because the data exchanged through the messages is buried in transport protocols that are inaccessible to the typical tester. Consequently, these defects are more costly to fix. The XML (eXtended Markup Language) schema is used to exchange messages via SOAP (Simple Object Access Protocol) are XML (eXtended Markup Language). The XML content contains many customized formats and fields. A simple mistake in format might lead to enormous set of permutations and error points and making it extremely hard to analysis the cause of error. Devising a strategy for testing that is comprehensive and efficient is a challenge.

2.3 Rapid Change and Dynamicity

The rapid change is the biggest risk in any SOA. A service-oriented system is built by abstract services. These abstract services get automatically bound to concrete services retrieved from the UDDI during run-time. Even though SOA is simple, flexible and agile its also complex, has multiple failure points and manageability issues. The whole deployment process needs to be well managed accordingly. The services that worked perfectly individually in the deployment environment might fail during integration testing when software is deployed into test environments. This is mainly because the developers tend focus on the component and miss the interaction of the component. Mostly smoke testing of a SOA implementation is similar to regression testing of the whole system.

2.4 Unavailability of Source Code and Structure of Services

The white-box testing approaches require knowledge of the structure of code and flow of data. However the users and system integrators utilize the services as just interfaces, making white-box testing very difficult. The information like the dependencies between service operations or the complete behavioural model of the service can be gathered from the service provider. In most cases the tester has to infer it by observing the service from outside.

2.5 Lack of Information about Integrated Components

Majority of integration testing effort is expended on creation of stubs. The source code unavailability makes production of stubs very difficult during integration testing. This makes the test effort and in turn cost of testing a service-oriented system more. The factors like cost, resource consumption and the undesired side effects must be gauged before using the services for testing. The dynamic and late binding of the services makes integration testing of service composition even more difficult. An invocation in a service composition might be dynamically bound to different end points.

2.6 Unanticipated Evolution of Participating Services

The services utilized in service-oriented systems might undergo changes. However, this is not notified to integrators/users accessing it. Hence as a consequence, those systems making use of the service might experience failure of the system or the SLA (Service Level Agreement) might not be met. This can happen even when the functional behaviour is preserved. The service integrator who is unaware of the changes might not be able to identify the source of a failure. When the changes are communicated, the service integrator still needs to perform a set of regression tests to ensure the proper functioning of the system.
2.7 Reliance on Service Providers

A service-oriented system is built mainly based on the behavioural model of the service and the QoS (Quality of Service) levels met by the service. This information is given by the service provider and is also used to comprehend the service characteristics, discover the service and to select it. However, there is no guarantee on the integrity of the information a service provider gives. In order to influence the selection decision of the service consumers the service provider may give incorrect/inaccurate description of a service’s functional and non-functional behaviour.

2.8 Lack of Effort and Funds

The effort and cost spent on testing is always more in any software application. The SOA implementations are always complex and has several combinations to be tested. Hence the effort needed to validate/test every possible workflow and condition in a SOA system is huge and hence very costly. The best way to reduce the cost is to pick the test cases that will validate most of the business critical paths of the software in the smallest amount of time. Moreover in any service-oriented architecture all the components must be tested in isolation and also validated for interdependencies.

2.9 Test Environment

One of the most crucial challenges to overcome is creation of test environment that is as similar as possible to the deployment environment. A test environment is used by testers to design, implement, execute and manage the testing process. Moreover it also enables capture of information about the functional and quality aspects of the items under test. The items under test in a SOA implementation can be the services, infrastructures and end-to-end business processes. Establishing a testing environment is difficult because the service-centric solution is widely distributed, with heterogeneous hardware and software (SOA infrastructure, services, operating systems and databases). Moreover mirroring capabilities of deployed environment in a testing environment is very costly. To reduce the cost few organisations develop testing environment that is nearly similar to deployment environment providing workarounds for the missing aspects. In some cases, test copies of deployment infrastructure may be available. There also cases when virtual machines are used mirror parts of the infrastructure.

Any level of SOA testing needs to develop new strategies, or to adapt existing ones, to deal with these challenges. A service-oriented system has to bring both business and technology agility. Testing service-oriented applications is not as simple as finding bugs within the developer’s code or on the screen of a given user interface. It is complicated and challenging due to the distributed and dynamic nature. Software quality processes for SOA system must evolve along with the architecture. The focus must be to genuinely test a business process and maintain context across the entire workflow. The white paper by iKTO [3] suggests three C’s that improves SOA testing [3]. Complete testing of business workflows across every heterogeneous technology layer of the SOA, at both a system and component level. The whole team involved in the SOA implementation must also collaboratively involve in quality assurance process. Both developers and testers together must create test cases that prove how SOA meets business requirements. As the services are consumed at runtime the SOA deployment should be validated continuously to ensure that business requirements will continue to be met as the system dynamically changes. Another challenge involves in deciding when to stop testing a SOA system. A sophisticated and powerful tool is necessary to test all the headless services in a typical architecture. Such a tool demands more investment
on test framework, either to build or to buy. The test framework should provide the visibility required to find the defects in the architecture. Business agility requires SOA services to be controlled by the business goals.

3. Elements to be Tested

The complete testing a service-oriented system involves validation of functional, non-functional and conformance characteristics of all the elements that are available for testing. The elements that build the system are the infrastructure components and the services. The end-to-end business sequence provides the business logic to perform an enterprise-level capability. However, the sequences have to be validated to assess the underlying business logic. Here the services might be services that invoke other services or composite services.

3.1 Infrastructure Components

An enterprise service bus (ESB) is a software infrastructure that acts as intermediary layer of middleware. ESB creates the process execution environment for the web services to exchange messages, route the content and integrate through XML-based data transformation. This enables a set of reusable business services widely available. It also increases connectivity and adds flexibility and controls important resources it binds. The mediation services are the services which provide a link between the application services and the ESB. Such may be partly or wholly implemented, depending on the breadth of actual requirements. The first step for any enterprise that wants to implement SOA is to evaluate the form of ESB that would be required. Typically the SOA infrastructure must provide capabilities to register and discover services, manage metadata, provide security, and exchange messages between services. Such capabilities might be achieved by using custom-developed software or commercial products. In some cases a combination of both custom and commercial products is used. In few cases even open source software is used. A typical SOA infrastructure consists of a registry from where the services are discovered, ESB, application and web servers, and databases. The SOA infrastructure capabilities provided by commercial vendors differ by several factors. The core elements that make up the infrastructure might differ. The tools used to support the infrastructure might also differ according to the vendors' business. Lastly, the capabilities such as repository and the services for authentication of access also depend on the vendors.

3.2 Web Services

As web service a software system designed to support interoperable machine-to-machine interaction over a network it is best suited for SOA implementations. Testing of web services includes testing basic web service functionality and interoperability at the same time validating the QoS agreed upon. Moreover, the service interface and its implementation along with message format and payload must be tested. Several times individual web services might not be able to encompass a business process in such cases multiple web services that interact to provide a business capability are composed together. Web service orchestration and web service choreography are the two types of composites. Web service choreography involves web service participants that interact with each other to communicate stateful data to direct behavior of the composition. The interactions between the services naturally accomplish a business goal. However, in orchestration the orchestrator service initiates the communication of one or more individual web services such that they accomplish a business goal. Similar to elements of SOA implementation the elements of a web service orchestration are the
composition code, the individual web services, and the infrastructure components used to realize the orchestration.

3.3 End-to-End Business Sequences

When the end-to-end business sequences are created role of end-user involved in the sequence must be taken into account. Moreover the back-end, front-end applications, databases and services involved in the execution of the sequence should also be considered [4]. Testing SOA end-to-end demands a great insight into what's happening inside the SOA environment. All the elements starting from the services, messages, interfaces to the business processes must be validated for correctness. The most effective way to validate a business process implemented using SOA is through continuous end-to-end testing process that inspects each layer of the SOA and infrastructure components as it's developed, run and maintained. Mostly end-to-end testing identifies defects that are related to flow of events pertaining to a set of conditions.

4. Testing Perspectives of the Stakeholders

There are several stakeholders who play different roles in developing an SOA implementation [3, 5]. The role and responsibilities of each stakeholder in testing the implementation must be defined in the test plan. This avoids unnecessary discrepancies in the issue resolution process.

4.1 Service Developer

Service developer builds the services. He delivers both the interface and implementation of services. He can build services either by using already existing component and customizing it as a service or by implementing from scratch. The developer performs white-box testing to validate the service’s non-functional properties and its ability to properly handle exceptions. The cost for testing is limited. As consumer and his real-time infrastructure are unknown the non-functional testing is unrealistic. Moreover the network configuration or load is unknown too. The service specification documents are used to generate test cases which might not reflect the real-time business scenario.

4.2 Service Provider

Service provider is the one who supplies services either by employing developers to build services or developing services. Thus sometimes a service developer might also be a service provider. The service provider focuses on providing services that meet the SLA agreed upon with the consumer. Testing costs are limited. Mostly a provider uses black box testing techniques. The service specification documents are the base documents used to generate test cases. Being unaware of the consumer’s infrastructure and network configuration or load the non-functional testing does not reflect the real time data. The service provider is responsible for setting up the environment for executing tests and analyzing results.

4.3 Service Integrator

Service integrator is one who utilizes existing services either to create composite services or to create an end user application. The existing services might be composite services too. The service integrator chooses service/s that suites and satisfies the functional and non-functional requirements made at design time. Testing from this perspective focuses on the business need rather than implementation detail which demands several service invocations.
This results in costs for the integrator and wasted resources for the provider. Runtime discovery and ultra-late binding can make this more challenging because the service selected is one of many possibilities. Furthermore, the integrator has no control over the service in use, which is subject to changes during its lifetime.

4.4 Service Consumer

Service consumer uses a service directly; it might be an atomic or composite service. In general the service consumer might be a person/organization that buys services to create service-oriented applications. However a service consumer might also be the end user. If the end user is the consumer then end-to-end testing is required else service level testing is done.

4.5 Infrastructure Provider

Infrastructure provider provides the Enterprise Service Bus (ESB) which used to connect the elements that build the SOA system. Moreover he also provides mechanisms to discover and register services in the Universal Description, Discovery and Integration (UDDI) to other stakeholders. Testing from this perspective is limited to UDDI and ESB alone.

4.6 Third-party Certifier

Third-party certifier potentially certifies whether a service (individual or composite) works as expected. Generally a third-party certifier is employed by the service integrator to validate a service’s fault-proneness. Thus the number of stakeholders and resources involved in testing activities is reduced. The certifier performs testing from the same network configuration as the service integrator. Moreover the certifier does not test a service from the perspective of the service integrator.

4.7 End-User

End-User uses the service-oriented systems that employ services. The user doesn’t know the technicalities of service-orientation or testing. However his only concern is that the system he uses works. Acceptance testing and beta testing require more cost and wastage of resources. Moreover the runtime discovery and ultra-late binding attributes of SOA makes this more challenging because the bound service is one of many possible or even unknown services.

5. Levels of Functional Testing

This section details the different levels of testing namely (i) unit testing of atomic services (ii) composition testing (iii) regression testing and (iv) end-to-end testing.

5.1 Unit Testing

Service is the heart of a service-oriented system, providing the main functionality. Validating whether each service meets its functional requirements is the first in SOA testing. Testing atomic services might be considered as component testing but the dynamic and adaptive nature of SOA makes it challenging. Unit testing can be white-box if the source code is available. During unit testing the number of service invocations is proportional to the cost of testing. Moreover testing must not produce side effects in the real world. TAXI [5,17,29] (Testing by Automatically generated XML Instances) is a tool created for generation of test cases from XML schema. It is also well-suited for performing service black-box testing. Bai
et al. [14] used services WSDLs (Web Service Description Language) interfaces to produce sequence of operations that needs to be tested based on dependencies existing between operations. Moreover the test generation algorithms use XML schema available in the WSDLs to generate test data. Generally the XML schema that defines the input parameters of any service operation must provide the complete information about the attributes like acceptable range, format, etc of the parameters. However, in practice, this is almost never done, thus the tester has to specify ranges/admissible values manually. However lack of source code availability and complex input formats defined in XML schema makes test data generation harder.

5.2 Composition Testing

The services can be composed by following two complementary views; choreography and orchestration. In Orchestration a central element controls the business logic and execution order of the interactions. In Choreography interactions may involve multiple parties and multiple sources, but each element of the process is autonomous and controls its own agenda. Thus service composition testing needs to be classified into choreography-based and orchestration-based testing.

5.2.1 Orchestration-based Testing: The orchestration view focuses workflow or computations followed by a single partner of a composite service. Orchestration is the most common approach and is used both within service composition and business processes. First the sequence of steps a process is defined along with the conditions and exceptions. The orchestrator is the central controller that then implements a sequence of actions or interactions. There are several orchestration languages. Some build orchestrator as structured flows or links describing unstructured flows or as a combination of structured and unstructured flows. Unit testing of composition might follow either white-box or black-box technique depending on how the orchestrator is built. Generally the integration testing technique followed Top-Down or Critical-Modules. Mayer and Laubke [16] have proposed a unit testing framework for performing white-box testing using Web Services Business Process Execution Language (WS-BPEL) specifications. Yuan et al. [11] created a tool that generates test cases based on control flow and data flow using model checking technique. The tool also supports white-box testing. Luang et al. in [6] proposed a process that requires the specification of properties and uses BLAST model checker to generate positive and negative test cases. The model checking technique is used to prove that the composition is outside of the standard specification of the orchestration.

5.2.2 Choreography-based Testing: In the choreography view the whole flow of interaction among all the services involved in the composition is exposed. Choreography is used in scenarios that have complex processes with many interacting parts like event-based or agent-based systems. The rules that govern the behavior of each individual participant in the process are determined first. Then the interaction of the individual pieces gives the overall behavior of the process based on following their own rules. Complex business processes which have many interactions among the activities use choreography for composition. Choreographies can be defined using two approaches. When the interactions (messages exchanged) between all the services are described in the perspective of composite service then it is called the global choreography. In the local choreography the messages exchanged by each service takes into account the temporal and logical dependencies among the messages it exchanges. At the unit level, choreography can only be conformance tested, since it is intended only to certify the capability of interaction of the services in the composition. Integration testing is
always Model-Based. A black box testing technique for testing local choreography is introduced in [7]. Generally the UDDI is used for registering the services. It can be used to validate the WS conformity to the published interface. The finite state machines are used to model the choreography of such interfaces. In the case of [7] the behaviour of a service is modelled as finite state machines, while in [8] component interfaces are described as graph grammars. Integration testing of local choreographies is done by checking the compatibility among two services. A different approach for generated black box unit tests is presented in [9]. The contracts agreed upon by the consumer and provider are transformed to graph grammars and then used for test case generation. The testing object oriented software testing techniques can be used for testing choreography [10, 11]. The limitation of using object orientation technique is that correlation and long running transactions aspects of service composition are not taken into account. Li et al. [19] proposed a testing technique that generates test cases from WS-BPEL specifications. This technique can be used for both orchestration and local choreography.

A service-oriented system results from a collection of services and infrastructures that when assembled appropriately will result in new end-user applications. Seamless interoperability is one of the key attributes of service orientation. However service provider cannot design services such that they can interoperate all the service consumers’ system. At the component level, interoperability means the proper interaction between the web services. This relies on agreement contracts accepted by both service consumer and service providers. There are several types of agreements at different levels that enable successful interoperation. Interoperability testing cannot be handled using traditional techniques that focuses on individual elements and not the interactions between them. The Service level agreements between the service provider and service consumer must be well understood to carry out effective interoperability testing. The web service standards address the interoperability check at the machine level and syntactic level. However, the standards that exist allow for testing services in isolation for their conformance to those standards. Mei et al. [20] proposed a method for data-flow testing using the BPEL. The XPath Rewriting Graphs (XRG) is created from the XPaths specified in the BPEL scripts. Then the CFG (Control Flow Graphs) extracted from WS-BPEL is combined with the XRG to create a new model named X-WSBPPEL. Based on variable definition and usages over XPaths a set of data-flow testing def-use criteria are coined. The data-flow testing is performed on the X-WSBPPEL models. Bartolini et al. [12] proposed a data-flow testing approach that converts the WS-BPEL process into CFGs. These CFGs are then used to generate test cases. The assumption of this approach is that data-flow coverage criteria conceived for traditional CFGs can be applied for data-flow testing of service compositions. [21-22] proposed an approach to map BPEL process, sub-activities (related to time) and elements to our formalism WS-TEFSM. This approach also supports active testing and passive testing. The timed test cases generated from WS-TEFSM are used for active testing and monitoring the test execution of a Web services composition is passive testing.

5.3 End-to-End Testing

End-to-end testing of a service-oriented system refers to testing all the business flows that constitute the business need for the implementation. The implementation might comprise of several applications and spread beyond the organization boundaries. Execution of a business flow might involve invocation of a number of services, usage of applications and receiving input from humans. The invoked SOA services may be provided by different organizations.

99
The following factors need to be considered for end-to-end testing:

- The distributed and adaptive nature of SOA demands a strong and flexible test environment. Moreover the test environment must also support the multiple service providers and disparate infrastructures. Mostly the business processes would be complex and long-running and hence test flows would also have several complex steps.

- Any change in the service/s invoked in an end-to-end business sequence may indicate a need for regression testing of the sequence. An efficient change management protocol has to be convened and agreed upon by the stakeholders.

The end-to-end testing of a service-oriented solution is similar to testing any other application with one exception: test sequences must be devised to validate the correctness of the business processes along with additional test sequences to validate the dynamically composed business process. The basic steps in end-to-end testing of a SOA application are:

Step 1: Identify business critical test scenarios

Any service-oriented system is complex and huge. Hence the number of end-to-end business sequences would be more. Validating all those end-to-end sequences would be time-consuming and in-effective. There are several manual and automated approaches to identify end-to-end sequences and establish test scenarios for SOA systems. In the manual approach the business requirements are analysed, then the requirement are broken down into events and the triggers associated with those events. The test cases are finally defined and built according to the events and triggers which correspond to functions and conditions. The following process steps, derived from Sikri [13] are representative of manual processes:

i) A tree structure of high-level business flows including inputs, outputs and execution paths is built.

ii) For each flow the post and pre conditions along with input data is identified.

iii) The test scenarios are created based on the flows and conditions.

iv) The test scenarios are then used to build tests cases.

The quality of service attributes must be related to appropriate scenarios. In some implementations the services that available online might be used. In such scenarios the runtime load, security and usage are unknown until the service is actually being used. In such cases an incremental approach is used to adjust the system to meet QoS goals. The automated approaches [23, 26, 32] to identify end-to-end sequences and establish test scenarios mostly use the business modelling languages like UML (Unified Modeling Language). Usage of the modeling languages in establishing test scenarios is still in maturing stage.

Step 2: Identifying Dynamic Composition Sequences

Research addressing testing of dynamic composition is a still maturing area. Dynamic composition must satisfy all the level of agreements between service provider and consumer. Thus far standards supporting such alignment are not yet available. Mostly the local conventions and standards are used. However those might be inconsistently defined and hard to enforce across organizations. All the dynamic compositions that are most likely is identified and tested. The impact of the composition on the network and SOA infrastructure is saved for future use. Moreover from functional aspects the impact on the individual services and other critical composition is also recorded.
Step 3: Establishing realistic test environment and data

Creation of realistic test data and environment has been a problem in testing any application and is for service-oriented applications too. It involves realistic simulation of the environment in terms of capabilities, data and loading. The service providers are liable to furnish details of the type of test data available and the guidelines about how to test their services. Several tools exist to simulate the data used in production environment. The test data used for validating QoS attributes must be highly reliable and similar to real time data. Moreover realistic loads are established to test the infrastructure components (network and SOA) and the invoked services/related applications.

Step 4: Reporting and Analyzing Test Results

Capturing test metrics in a service-oriented application is very challenging due to the dynamic nature of the implementation. The format of any metric data depends on the characteristics of the infrastructure and tools used for construction. A special service can be used to capture and categorise the data to make specific decisions after analysing the data. There are several tools that make test management easier. The test results collated can be consolidated and analysed using those tools. This provides a single point of review. There are also testing tools that are part of both design and development suite. This enables linking of test cases and scenarios to requirements, ensuring that the notion of requirements traceability is satisfied.

5.4 Regression Testing

This section focuses on the specific aspects of regression testing that need to be considered in an SOA environment. Regression testing involves testing and identifying bug fixes to ensure that defects have not been introduced as a result of evolution. Regression testing of any end-to-end business sequence is triggered whenever changes happen in any one of the following areas;

- Life-cycle management of components
- Service composition/bindings
- Deployment configurations
- Back-end systems or data sources
- Service’s functional behavior
- Service’s non-functional behavior
- Requestor/responder application
- Infrastructure and/or services invoked
- Back-end systems upgrades to Service Contracts (WSDL)

Ruth and Tu [24, 25] proposed a regression test suite selection technique that uses CFGs that should be able to highlight the changes that can trigger regression testing. Ricca et al. [14-16] proposed a framework for integrated testing to assess the relevance of acceptance test cases that can be used to validate the business requirements. The test suite is periodically used to re-test the service. [27] proposed a test data generation method using the contracts agreed upon by the service provider and consumer. The contracts and OWL-S models are used to create Petri nets that are used for test data generation.
6. Recommendations

The previous sections analysed the various test techniques applied at different levels of SOA testing. This paper also presents few general recommendations for testing SOA applications. A strong SOA test strategy, which accounts for SOA infrastructure, services (individual and composites) and end-to-end testing must be developed. The test methodology for any SOA application must take into consideration the roles and responsibilities of each stakeholder. When testing service compositions the approaches used for building the composite service must also be considered. An approach to identify business critical test sequences and dynamic compositions has to be convened. The trigger for regression testing of services, compositions and end-to-end testing has to be identified. Any test framework would be effective and efficient if it can improve the test coverage in less cost, number of testers, resources utilization and rework. The most effective way to validate SOA business processes is by performing continuous end-to-end testing process that checks each layer of the SOA when it is developed, run and maintained. A proper test plan and strategy would aid in efficient and effective quality assurance through model based testing of even complex service-oriented systems.

7. Conclusion

Software testing is most expensive and crucial activity in the life-cycle of any software system. The dynamic nature, automated service discovery and ultra-late binding make SOA testing very challenging. Various existing testing techniques at different levels of testing have been presented in this paper. The paper also presents ways to improve testability of any service-oriented application by few proposals. There is a need for devising an efficient strategy for testing the SOA infrastructure, web services (individual and composite services) and end-to-end business sequences. A good strategy can make the testers responsible for testing a service-oriented application comprehends the business processes easily and effectively.

References

Authors

**Poonkavithai Kalamegam** received B.Tech degree in Computer Science and Engineering from Pondicherry University and M.E. degree in Computer Science from Anna University. Her research interest includes Service Oriented Architecture, Model Based Testing, and Web Service Composition Testing. She has around 10 years of industry experience in functional testing of banking domain applications. She has been involved in all phases of testing, starting from estimation for testing phase to closure with test summary report. She has worked in Cognizant Technology Solutions for 6 years. JP Morgan Chase Bank, Dutsche Bank and Boeing Financials are some of the clients she has worked for. She is currently pursuing PhD degree in the area Web Service Composition Testing.

**Dr. G. Zayaraz** is currently working as Associate Professor in Computer Science & Engineering at Pondicherry Engineering College, Puducherry, India. He received his Bachelor’s, Masters and Doctorate degree in Computer Science & Engineering from Pondicherry University. He has published more than Thirty five research papers in reputed International Journals and Conferences. His areas of specialization include Software Architecture and Information Security. He is a reviewer/editorial member for several reputed International Journals and Conferences and Life Member of CSE, and ISTE.