An External Quality Supporting Test- Driven Development of Web Service Choreographies

Pogiri Ravi Kumar
M.Tech Student
GIET
Rajahmundry, A.P, India

P.V.G.K.Jagannadha Raju
Professor
GIET
Rajahmundry, A.P, India

S.Maruthuperumal, Ph.D.
Professor and HOD CSE and IT
GIET
Rajahmundry, A.P, India

ABSTRACT
Recently, software development teams using agile processes have started widely adopting test-driven development. Despite its name, "test driven" or "test first" development isn't really a testing technique. Also known as test-driven design, TDD works like this: For each small bit of functionality the programmers code, they first write unit tests. Then they write the code that makes those unit tests pass. This forces the programmer to think about many aspects of the feature before coding it. It also provides a safety net of tests that the programmers can run with each update to the code, ensuring that refactored, updated, or new code doesn't break existing functionality. TDD can also extend beyond the unit or "developer facing" test. Many teams, including my own, use "customer facing" or "story" tests to help drive coding. These tests and examples, written in a form understandable to both business and technical teams, illustrate requirements and business rules. Customer-facing tests might include functional, system, end-to-end, performance, security, and usability tests. Programmers write code to make these tests pass, which shows the product owners and stakeholders that the delivered code meets their expectations. The results indicate that, in general, TDD has a small positive effect on quality but little to no discernible effect on productivity. However, subgroup analysis has found both the quality improvement and the productivity drop to be much larger in industrial studies in comparison with academic studies. A larger drop of productivity was found in studies where the difference in test effort between the TDD and the control group's process was significant. A larger improvement in quality was also found in the academic studies when the difference in test effort is substantial; however, no conclusion could be derived regarding the industrial studies due to the lack of data. Finally, the influence of developer experience and task size as moderator variables was investigated, and a statistically significant positive correlation was found between task size and the magnitude of the improvement in quality. Choreographies have been proposed as decentralized and scalable solutions for composing web services. Nevertheless, inherent characteristics of SOA such as dynamicity, scale, and governance issues make the automated testing of choreographies difficult. Nevertheless, inherent characteristics of SOA such as dynamicity, scale, and governance issues make the automated testing of choreographies difficult. The goal of our research is to adapt the automated testing techniques used by the Agile Software Development community to the SOA context. To achieve that, we aim to develop software tools and a methodology to enable Test-Driven Development (TDD) of web service choreographies.

Keywords
Test-driven development,meta-analysis, code quality, programmer productivity, agile software development.

1. INTRODUCTION
Service-Oriented Computing has been considered the new generation of distributed computing, being widely adopted. Service- Oriented Architecture (SOA) aims at the implementation of Service-Oriented Computing by using web services as the building block of applications. Computability of services is one of the SOA principles, however, few approaches for composing services have been proposed. Orchestration is a centralizedapproach for service composition. Although straightforward and simple, its centralized nature leads to scalability and fault-tolerance problems. To face this problem, choreographies of web services have been proposed as a decentralized scalable composition solution. In spite of all the benefits and advantages of web service compositions, the automated testing of composed services has not yet received the needed attention. There are few techniques and tools directly applicable for testing these systems because of the dynamic and adaptive nature of SOA. Some tools, such as SoapUI1 and WebInject2 have been developed for testing atomic services. Since composed services are accessible as atomic services (from the user perspective), these tools can be used in larger scopes. Nevertheless, on such approach, both orchestration and choreography are taken as black-boxes, preventing the use of testing strategies such as unit and integration tests. In the unit testing approach each service participating in a composition is taken as a unit, while on the integration testing approach, the interaction among these services must be exercised and verified. TEST-DRIVEN Development (TDD) is among the cornerstone practices of the Extreme Programming (XP) development process and today is being widely adopted in industry both as part of a large-scale adoption of XP and as a stand-alone practice. TDD is commonly considered to be the amalgamation of test-first development, in which unit tests are written before the implementation code needed to pass those tests, and refactoring, which includes restructuring a piece of code that passes the tests in order to reduce its complexity and improve its clarity, understandability, extendibility, and/or maintainability. TDD is often described with the so-called "redgreen- refactor cycle" that consists of the following steps:

- Design and add a test.
- Run all tests and see the new one fail (red).
- Add enough implementation code to satisfy the new test.
- Run all tests, repeat 3 if necessary until all tests pass (green).
- Occasionally refactor to improve code structure.
- Run all tests after refactoring to ensure all tests pass.

The use of TDD is claimed to bring improvements in code quality and productivity. However, research studies investigating the effectiveness of TDD have failed to produce conclusive results; in fact, all possible outcomes—positive, negative, and neutral—have been reported for both quality and productivity improvements obtained with TDD.

2. METHOGLOGY

Since we are also interested in testing the components of choreographies, i.e., individual services, we started studying the existing software tools for automated testing of atomic services. Soap UI is developed in Java and provides mechanisms for functional, regression, and performance tests. From a valid Web Service Description Language (WSDL) specification, the Soap UI tool provides features to build automatically a suite of unit tests for each operation and a mock service to simulate the web service under testing. It also provides mechanisms to measure test coverage. Due to the distributed and dynamic nature of orchestrations and choreographies, there are yet few tools for testing and monitoring the services participating on such compositions. BPEL Unit provides mechanisms for specifying, organizing, and executing tests for a Business Process Execution Language (BPEL) process. Its goal is to exercise the internal behavior of such processes, validating its outputs by predefined inputs. In the context of choreographies, there are even fewer tools than for orchestrations. Pi4SOA3 is a software tool for modeling choreographies in WS-CDL by producing the global model and, then, a BPEL specification for each participant, describing their role in the choreography. Once modeled, it is possible to validate the flow among the web services by simulation. This way, Pi4SOA provides design time mechanisms to verify the global model specified in WSCDL. An initial effort in understanding the current scenario of testing techniques for orchestrations and choreographies was conducted by Bucchiarone. Later, a more comprehensive survey covering SOA testing was conducted by Canfora and Di Penta . Both works discuss and present alternatives for testing web service compositions based on testing strategies applied to traditional client/server systems. Acceptance testing aims at verifying the behavior of the entire system or a complete functionality. It can be performed by taking the composition as an atomic service. In this situation, black-box tests and tools that can be applied are equivalent to atomic services. In the unit testing approach, each participant is a unit to be tested. For choreographies, the expected behavior for each partner is defined by its role in the choreography. Thus, black-box techniques can be applied for validating this behavior against this specification role. In the integration testing approach,the interaction among components (services must be exercised and verified. Nevertheless, the lack of information about certain partners and the impossibility of exercising some third-party services prevent the integration tests. In the SOA context, through the dynamic binding property, the endpoints of a participating service are chosen dynamically. Such property can raise the integration test costs since strong criteria might require testing all possible endpoints. Model-Based Testing (MBT) can be an alternative to derive integration test cases. MBT refers to an approach to derive test cases from the exploitation of formal models. Some works in this direction try to derive test cases automatically from choreography specifications, applying algorithms defined for conformance checking. Some tools have been developed to convert choreography models into UML diagrams, and then, derive test cases from these diagrams. Zhou et al. have proposed a new approach for the validation of the choreography model by checking a global model written in WS-CDL to ensure the quality of its design. First, the choreography is parsed into a data-object graph. Then, through relational calculus, static validations are applied. The meta-analysis procedure has gained considerable attention in recent years as one of the effective ways to quantitatively summarize and, if possible, interpret the results of a collection of single studies on a given topic. The analysis proceeds through a number of distinct steps, as follows:

2.1 Study Identification and Selection

The identification and selection process proceeded in three stages. First, we identified candidate studies by querying the electronic databases of the ACM Digital Library, IEEE Xplore, Springer Link, ISI Web of Science, and Scopus, using the strings "Test Driven Development," "Test First Development," and "TDD" to search through the Article Title, Abstract, and Keyword fields. The generated matches were filtered to include only studies published in peer-reviewed journals or proceedings from peer-reviewed conferences. The resulting matches were prescreened for relevance by reading through the titles and abstracts but also, in some cases, going through the introduction. All studies found to be relevant, as well as those whose relevance was still unclear, were selected for a more thorough analysis. In the final stage, each of the authors read all of the selected studies and individually compiled a list of studies to be included in the review. The individual lists were then compared and all differences were resolved through discussion. Accordingly, a final list of studies was derived which would form the subject of the upcoming meta-analysis.

2.2 Data Extraction and Output Categories

The data extracted from the studies was classified into three categories: Context, Rigor, and Outputs. Attributes in the first category recorded contextual and other high- level details regarding the studies, including The authors of the study, the number of participants, and the context—academia or industry—in which the experiment was conducted. Attributes in the Rigor category aimed to help assess the extent of the applicability of a study's results according to the criteria for study rigor described in. These attributes include the following: .CT, which indicates the manner in which testing was done by the control group— iteratively, i.e., interleaved with coding. OA, indicating the other agile practices that were included in the development processes; .development and programming experience of the subjects; .task size of the final application (in LOC); .duration of the project; .information about process conformance in the target group (i.e., adherence to the widely accepted principles of TDD development); .details of training received by the subjects prior to the experiment.

2.3 Inclusion and Exclusion Criteria

Studies were included in this meta-analysis if they reported results on one or more experiments in which the effectiveness of TDD was compared with that of a more traditional (i.e., Test-Last) approach. Such experiments were designed with subjects being divided into two or more groups, each of which developed the same or similar products with at least one group following either development approach. Studies were only included if they reported quantitative data on at least one of the investigated outcome constructs. The use of other agile practices along with TDD was not considered as a limiting factor.

2.4 Standardized Analysis

All standardized effect sizes in this paper were computed using the Comprehensive Meta-Analysis V2 tool by BioStat, Inc. The Hedges' g statistic was chosen as the standardized effect size measure for the analysis as it exhibits better characteristics for smaller samples when adjusted for small sample bias in comparison with other parametric measures such as Cohen's d and Glass' Delta. The Hedges' g statistic is calculated as g¼ g = mt –mc/spooled; where mt and mc refer to the mean values reported for the treatment and control groups, respectively, and spooled refers to the pooled standard deviation.

3. SOFTWARE PROTOTYPE

Our prototype consists of ad hoc bash scripts for a choreography enactment, JUnit test cases for automated testing of the running choreography, and a user interaction prompt for executing the scripts and tests. In this section, we first present and explain the choreography developed and then, we present our automated test scripts and approaches for applying unit, integration, and acceptance tests on the running choreography

3.1 The Tested Choreography

To validate our prototype, we designed and implemented a simple choreography for booking a trip on OK (Open-Knowledge). The choreography participants were essentially SOAP/WSDL services and RESTful web services. A user plans to take a trip and informs the traveler service where and when to go. After ordering a trip through this choreography, the user can reserve an e-ticket, and finally, confirm (book) or cancel it. Initially, traveler invokes travel agency, which searches for the required flight on the airline. After selecting a flight, traveler requests a trip reservation to travel agency, which requests a flight reservation to the airline. After these two interactions, a user can request the traveler to cancel the reservation or to book it. Since this process of booking the trip is more complex than the previous one. The process of booking a trip starts with the user requesting this operation to the traveler service, which in Figure 1 is represented by a white envelope. Then, the traveler service makes a book trip request to the travel agency, which calls the acquirer to check whether the user can afford the flight and its services or not. The acquirer service notifies the purchase refusal to the travel agency and airline services if the user cannot afford the trip. In this case, these services send messages to the traveler reporting the refusal. Otherwise, the acquirer service sends a payment confirmation to the travel agency and airline services. After that, the airline service confirms the flight price with the travel agency and sends the e-ticket to the traveler. After receiving the confirmation from the acquirer and the airline, the travel agency sends to the traveler a report (statement) with the total price paid. Finally, the traveler sends this response to the use Implemented Test cases We developed automated test cases for applying the studied techniques and strategies on our choreography. All tests were developed using the JUnit framework and can be automatically compiled and executed by our software prototype.

1)Unit tests: In choreography context, services are considered the units for unit testing. Thus, our unit tests validate the service behavior by verifying each provided functionality. In our current prototype, to test SOAP web services, a Java SOAP client (developed using JAX-WS6) needs to be developed for each service endpoint (i.e., the client is specific for each endpoint). Once developed, the tests use this client to invoke the services. Thanks to the inherent flexibility of RESTful services, we developed a generic REST client (i.e., it is not restricted to a specific endpoint).

2) Acceptance tests: Differently from other testing strategies, acceptance tests verify the behavior of the entire system or complete functionality. From the point of view of an enduser, the choreography is available as an atomic service. Thus, the acceptance test validates the choreography as a unit service, testing a complete functionality. In this context, this type of test is similar to the approaches of unit testing using the black-box model, and there is no need to know how the service is implemented. On our approach, a developer specifies the tests by calling a service that activates the choreography. Before the execution, the developer needs to execute a script that enacts the choreography and deploys the services. Then, the tests are hacking). Since SoapUI does not provide support for executed and the actual results are compared with the expected output values. In the choreography example explained above, the traveler peer is the service that triggers it. Therefore, to test the Order Trip Operation, the developer calls the method on the traveler web service and compares the returned object properties with the expected ones.

3) Integration tests: Integration tests intend to solve the problems found when unit tested components are integrated. Their goal is to verify the unit interfaces and interactions among system components. Based on the discussion presented by Bucchiarone for integration testing of choreographies, we defined an approach for applying integration tests. After all services have been tested at the unit level, the approach focuses on integrating each service at a time in the choreography. Once a service is integrated, the choreography is enacted by the developer. Then, using runtime monitoring of the choreography, the framework verifies Whether the service just integrated behaves as expected. This step is achieved by checking the messages sent by that component. For each message, its name, destination, and content are compared with the expected values.

4. QUANTITATIVE ASSESSMENT

As described previously, our integration testing approach must collect the messages exchanged among the services. Such procedure might cause an overhead in the choreography execution. We have conducted a quantitative assessment to evaluate possible overheads. In this assessment, we first deploy our choreography example (see Section III-A) on a cluster. Each service choreography was allocated on a dedicated node with a Pentium 4, 3.00GHz processor, with 1GB of RAM, running GNU/Linux and connected to a 100Mb/s LAN. The goal of this assessment was to compare the execution time of a choreography functionality using and not using ourapproach for monitoring the choreography at runtime. We have chosen the order trip functionality for the experiment. In this execution, 4 messages are exchanged among the services. We measured the execution time of 1, 2, 4, 8, and 16 sequential order trip executions, collecting and not collecting all the 4 messages exchanged. First, each sequence was executed 30 times, e.g., in the case of 8-sequential order trip executions, we had 30 samples, each one with the 8-sequential executions. Then, we extracted the average and standard deviation of these samples.

5. QUALITATIVE ASSESSMENT

Soap UI provides functionalities for the automatic generation and execution of test cases from a valid URI; but the tester must still fill in the XML-Soap envelope, which can be cumbersome. As depicted in Figure 2, on our unit tests, the tester interacts with the web services under testing in an object- oriented way, by invoking methods instead of manipulating XML-Soap envelopes. However, our web service client generation is not fully automatic. In our ongoing work we are seeking to combine the automatic client generation of SoapUI with the high- level, object-oriented testing scheme of our approach (free of the burdens of XML integration tests, we developed a new approach (described in Section III-B3). To illustrate the benefits of our approach, we implemented a use case based on our choreography example. Considering that the travel agency can search for flights in more than one airline, suppose that another airline service is integrated into the choreography. This new service is from a Brazilian provider, and consequently, it charges all tickets in BRL (Brazilian Real), but our choreography only works with USD (United States Dollar). Initially, all unit tests for this new component pass and the incompatible currency is not noticed at this stage. Then, the integration test detects that the acquirer service charged theticket price incorrectly since its service does not apply currency conversions. In this example, our approach reveals the error and points to where, in the choreography, it could be fixed. To correct the error, one must add, for instance, a currency converter service between the travel agency and acquirer services. In the absence of our approach, one is limited to acceptance testing strategies to validate a service integration. In this case, the choreography is taken as a black-box, preventing the tester to discover where exactly the error occurred. With our approach, after identifying a problem, we can start collecting and analyzing the messages exchanged to isolate the problem. In our current prototype, for collecting a specific message, the whole choreography functionality flow must be performed. Thus, our integration tests take, at least, the same time that an acceptance test for that choreography flow takes. To improve our approach, we intend to develop a mechanism that can stop the choreography after collecting the desired message.

6. MODERATOR VARIABLES

The above discussion on the outcome constructs has shown that differences in effect sizes of subgroups are most likely due to other variables. Two among the most important variables are developer experience and task size, as highlighted in the discussions in the Academic versus Industrial subgrouping. In this section, we take a closer look at the moderating effects of these variables, beginning with a brief summary of previous research that documents the impact of these variables on performance of TDD-based development process.

6.1 Developer Experience

Although the success of TDD is dependent on skills in a number of different areas, including programming, testing, design, refactoring, and thinking in a TDD style programming experience and exposure to TDD are the only variables that have been explicitly studied summarizes existing studies that relate the impact of experience on TDD. Mu¨ ller and Ho¨ fer compared the performance of finalyear undergraduate students from an XP course with that of professionals with at least five years of industrial programming experience. The latter group was also more experienced with regard to use of automated testing tools and exposure to TDD. All subjects individually developed a Java-based elevator control system following the TDD approach until they felt they were done; their programs were then evaluated using previously prepared acceptance tests. The subjects in the professional group were found to finish the task in shorter time, and this result was statistically significant. The difference was attributed to faster coding speed and higher level of programming experience. However, a larger proportion of programs prepared by the students passed the acceptance tests, but this result was not statistically significant. This somewhat unexpected result was attributed to the professionals' perception of the acceptance testing process as a regular adjunct to testing and thus being assigned lower priority. On the other hand, subjects in the students group viewed the acceptance tests as a more formal assessment criterion, and thus ensured, to a greater degree, that the implemented functionality was in working order prior to submission.

6.2 Task Size

We have also analyzed the relationship between task size and the magnitude of the improvement brought about by TDD; 14 data points (effect sizes and respective task sizes) could be obtained from the analysis on quality and 13 from the analysis on productivity versus task size; for clarity, the x-axis uses a logarithmic scale. A visible trend can be identified in the plot for quality, unlike the plot for productivity where such a trend cannot be easily observed; the relationships of both quality and productivity improvements with task size appear to be logarithmic in nature.

7. THREATS TO VALIDITY

The major obstacle in conducting this analysis was the lack of data available for computing the standardized effect size in each experiment. Although we partially overcame this obstacle by using an unstandardized effect size measure, all unstandardized measures within the context of this research suffer from two principal disadvantages.

8. CONCLUSION

This research intended to investigate the effectiveness of TDD by applying meta- analytical techniques to previous empirical research on the external quality and productivity outcome constructs. Despite Consider able differences among the experiments, valuable insight can be gained from this analysis. Overall, our analysis suggests that TDD results in a small improvement in quality but results on productivity are inconclusive. To gain deeper insight into the differing levels of improvement observed for both outcome constructs, we have also conducted subgroup analyses using two major subgrouping strategies. Under the Academic versus Industrial subgrouping, much larger improvements in quality were found in the Industrial experiments, which may be attributed to higher developer experience and much larger task sizes in those studies. Although the analysis on moderator variables identified a correlation with task size, no concrete evidence was found relating the experience level to the magnitude of the improvement in quality. one of our goals is to develop a TDD methodology that will help developers and project leaders to deal with the key-issues involved in testing large scale, distributed, Internet systems and will guide them in the production of effective and efficient test suites for web service choreographies. To achieve this goal, we first intend to develop an open source testing environment to support the methodology proposed. Based on the results of the current

work and on the lessons learned from the prototype development, we can derive some requirements and challenges that must be faced to achieve our future goals.

9. REFERENCES

[1] H. M. A. Bucchiarone and F. Severoni. Testing service composition. In 8th Argentine Symposiumon Software Engineering (ASSE'07), Argentina, 2007.

[2] G. Canfora and M. Di Penta. Testing services and service-centric systems: challengesandopportunities.IT Professional, 8(2):10 –17, march-april 2006.

[3] G. Canfora and M. Di Penta. Service- oriented architectures testing: A survey. In A. De Lucia and F. Ferrucci, editors, Software Engineering, volume 5413 of LNCS, pages 78–105. Springer, 2009.

[4] L. Frantzen, M. N. Huerta, Z. G. Kiss, and T. Wallet. On-The-Fly Model- Based Testing of Web Services with Jambition. In 5th International Workshop on Web Services and Formal Methods – WS-FM 2008, 2009.

[5] P. Mayer and D. L¨ubke. Towards a BPEL unit testing framework. In Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications, TAV-WEB '06, pages 33–42, New York, NY, USA, 2006. ACM.

[6] A. Stefanescu, S. Wieczorek, and A. Kirshin. MBT4Chor: A model-based testing approach for service choreographies. In Proceedings of the 5th European Conference on Model Driven Architecture – Foundations and Applications, ECMDA-FA '09, Berlin, Heidelberg, 2009.

[7] Z. Wang, L. Zhou, Y. Zhao, J. Ping, H. Xiao, G. Pu, and H. Zhu. Web services choreography validation. Service Oriented Computing Applications, 4, December 2010.

[8] L. Zhou, J. Ping, H. Xiao, Z. Wang, G. Pu, and Z. Ding. Automatically testing web services choreography with assertions. In Proceedings of the 12th international conference on Formal engineering methods and software engineering, ICFEM'10, pages 138–154. Springer-Verlag, 2010.

[9] T. Huedo-Medina, J. Sa´nchez-Meca, F. Marı´n-Martı´nez, and J. Botella, "Assessing Heterogeneity in Meta-Analysis: Q statistic or I2 Index?" Psychological Methods, vol. 11, no. 2, pp. 193-206, 2006.

[10] G. Melnik and F. Maurer, "A Cross-ProgramInvestigationofStudents' Perceptions of Agile Methods," Proc. 27th Int'l Conf. Software Eng., pp. 481-488, 2005.

[11] S. Kollanus and V.Isomo¨ tto¨nen, Understanding TDD in Academic Environment: Experiences from Two Experiments," Proc. Eighth Int'l Conf. Computing Education Research, pp. 25-31, 2008.

[12] A. Rendell, "Effective and Pragmatic Test Driven Development," Proc. AGILE, pp. 298-303, 2008.

[13] J. Langr, "Evolution of Test and Code via Test-First Design," Proc. ACM Conf. Object-Oriented Programming, Systems, Languages, and Applications, http:/www .objectmentor.com /resources/arti cles/tfd.pdf, Oct.2001.

[14] D.H. Steinberg, "The Effect of Unit Tests on Entry Points, Coupling and Cohesion in an Introductory Java Programming Course," Proc. XP Universe, Oct. 2001.

[15] J. Sanchez, L. Williams, and E. Maximilien, "On the Sustained Use of a Test-Driven Development Practice at IBM," Proc. AGILE, pp. 5-14, 2007.

[16] L. Madeyski, "The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design—An Experiment," Proc. Seventh Int'l Conf. Product-Focused Software Process Improvement, pp. 278-289, 2006.