Delft University of Technology
Software Engineering Research Group
Technical Report Series

Maintenance Research in SOA Towards a Standard Case Study

Tiago Espinha, Cuiting Chen, Andy Zaidman, Hans-Gerhard Gross

Report TUD-SERG-2012-001

TU Delft

SERG

TUD-SERG-2012-001

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Paper accepted at the European Conference on Software Maintenance and Reengineering (CSMR 2012)

Maintenance Research in SOA — Towards a Standard Case Study —

Tiago Espinha, Cuiting Chen, Andy Zaidman, Hans-Gerhard Gross
Delft University of Technology
The Netherlands
{t.a.espinha, cuiting.chen, a.e.zaidman, h.g.gross}@tudelft.nl

***Abstract*—Maintenance research in the context of Service Oriented Architecture (SOA) is currently lacking a suitable standard case study that can be used by scientists in order to develop and assess their research ideas, and for comparison, and benchmarking purposes. It is also well established in different fields that having such a standard case study system brings many benefits, in that it helps determine which approaches work best for specific problems.**

For this reason, we decided to build upon an existing open-source system and make it available for other researchers to use. This system, Spicy Stonehenge, provides many advantages for carrying out maintenance research: it is complex, extensible and, most importantly, openly available for anyone to use and build upon.

With this paper, we introduce Spicy Stonehenge as the standard case study SOA system, and we also present our research vision in the field of maintenance, reengineering, evolution and testing of SOA systems, and how these goals fit together with Spicy Stonehenge.

I. Introduction

While the actual term Service-Oriented Architecture (or SOA) was first coined in the mid 1990's by Gartner [1], [2], the ideas behind it, namely building software systems that are composed out of loosely coupled, interoperable components or services, goes back further. It was, however, the technology of *web services*, launched in 2000 as a set of standards to allow computers to communicate with each other [2], that acted as a catalyst for both industry and academia to really start investigating the possibilities of Service-Oriented Architectures. In particular, SOAs promise to (1) allow businesses to be more flexible as business needs change and (2) ease evolution due to the loosely coupled nature of the system [3].

When looking at the past decade of research in service-orientation, we can observe that although a lot of fruitful research has been carried out (e.g., see [4], [5]), many of the research efforts are isolated in nature. While this isolation is not bad per se, it does hinder progress. Symptomatic of the isolated nature of research in this area is the absence of a common case study that can be used as a benchmark. Indeed, Sim et al. report that benchmarking, when embraced by a community, has a strong positive effect on the scientific maturity of a discipline [6]. In particular, it allows to easily compare solutions and to perform replication studies. In many fields of software engineering, researchers have resorted to benchmarking in order to compare approaches and ultimately advance the field. Prime examples being the aspect-mining community that settled on *JHotDraw* as a standard case study [7], or the refactoring community that introduced the *LAN simulation* [8].

In order to unify the SOA community around a single case study, we propose a system that is at the same time realistic, easy to understand and which most researchers should be able to use as a "standard case study system". The system we propose — Spicy Stonehenge — is based on Apache Stonehenge and consists of an application composed out of several web services. The open-source nature of Spicy Stonehenge and its availability should stimulate researchers in the area of SOA, that normally resort to small examples built specifically for the context of their research, to choose for Spicy Stonehenge, thus enabling the benchmarking process that the community needs.

This paper makes the following contributions:

- We introduce Spicy Stonehenge, an open-source service-based Java software system as a possible standard case study for researchers working in the area of SOA.
- A brief survey of existing research initiatives in the area of SOA from which we extract criteria that need to be specified when performing a case study in order to allow future comparison and/or replication.
- A research agenda for online evolution and online testing in the area of SOA.

This paper is structured as follows: in Section II we present a background of similar research being done in this field, Section III describes our service-oriented system (Apache/Spicy Stonehenge) in detail, Section IV provides an overview of our future research agenda and lastly, Section V presents a summary of what is discussed in this paper.

II. Background Research

In our reconnaissance of the research area of Service-Oriented Architectures, we noticed that there is no standard case study being used by researchers. Furthermore, during our exploration of the field we also got the impression that a wide variety of small and/or closed source systems were being used as case studies for evaluating the research. In order to get a better feeling of how research in the area of SOA is conducted, we have performed a small literature survey where we specifically focused on the software systems that are being

TABLE I
SELECTED SOA RESEARCH PAPERS WITH CASE STUDIES

Paper	Description	Complexity	Impl. Tech.	Availability
[9]	Runtime monitoring of web service compositions	3 web services	Unknown	No
[10]	Assumption-based composition and monitoring of WS	1 web service	Unknown	No
[11]	Monitoring functionality of conversational services	1 web service with 3 interfaces	Unknown	No
[12]	Assessing the performance impact of service monitoring	1 web service	Unknown	No
[13]	Synthesis and composition of web services	2 services: AECS[1] and MPS[2] e-payment service	Unknown	Industry, API avail.
[14]	Exception handling for web service orchestration	3 web services	Unknown	No
[15]	Dynamic monitoring service compositions	Unknown	Unknown	No
[16]	Runtime monitoring requirements for service composition	Unknown	Unknown	No
[17]	Monitoring process for SOA	Unknown, KIM[3] project	Unknown	No
[18]	Adopting and evaluating SOA in industry	700+ services	J2EE, IBM WebSphere etc.	Industry case
[19]	Transformation-driven evolution for SOA	Unknown	Unknown	No
[20]	A case study in SOA-based platform design	120+ services	Apache CXF etc.	No
[21]	Automic sysnthesis for composable web services	1 service: AECS[1]	Unknown	Industry, API avail.
[22]	Ensure interoperability for service-oriented systems	del.icio.us[4]and OpenSocial[5]	Unknown	Industry, API avail.

used in case study research.

In order to characterize the case study systems being used in SOA research, we compiled Table I, which represents a small subset of research papers in the area of SOA. The papers that we selected for this overview originate from:

- The state-of-the-art report on service monitoring from the European S-Cube[6] project on software services [5]. We selected this survey because our research goals are aligned with many of the papers mentioned in this report.
- A selection of recent papers published at venues like CSMR, ICSE and ESEC/FSE, from which we expect a thorough validation.

The 14 papers listed in Table I are all representatives of case study research [23]. We now list some of our observations:

Self-created case study systems. From this selection of papers we noticed that some authors created their own simple non-industrial examples as case systems, which contain a very small number of services, e.g., [11] and [14] have one and three services respectively. It is arguable whether these small case study systems are representative of real service-based software systems. Some self-created systems also appear more complex. For example, Baresi et al. [15] describe an IT certification system which gives enrolled students a chance to try a certification test for free. However, the paper only describes the conceptual details of the system.

An important issue with self-created systems is that their set-up might be favoring the researched technology, which becomes extra hard to verify when these self-created systems are not publicly available. Looking at Table I we see that unfortunately, almost all systems are not publicly available.

[1]AECS: Amazon E-Commerce Services — http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

[2]MPS: Monte dei Paschi di Siena Group(an important Italian financial Group) — http://www.mps.it/

[3]KIM project — http://kim.cio.kit.edu/

[4]del.icio.us — http://delicious.com/help/api

[5]OpenSocial — http://code.google.com/apis/opensocial/docs/0.7/reference

[6]S-Cube — http://www.s-cube-network.eu

Closed-source systems. Some researchers are cooperating with industry and have the chance to get a real-world system as their research vehicle. For example, Momm et al. [17] apply their approach to a practical scenario developed in a project aiming to redesign a university's business process; Nasr et al. [18] provide an industry case study supported by a business service IT company. Also, in the paper by Pistore et al. [10], the authors mention that their approach was applied to some real applications, but no more details are provided.

Industrial case studies are extremely important in software engineering research, however, due to the closed-source nature of these software systems they cannot be obtained by other researchers. This means their results cannot be reproduced or compared, which strengthens our call for a common case study to compare techniques on.

Implementation technology. During this survey, we also focus on investigating the implementation technologies used in those case study systems, such as the programming language, the underlying frameworks, the communication protocols, etc. These pieces of information are necessary in different situations, e.g., (1) when practitioners want to use the experimental results and want to verify whether the results are applicable in specific circumstances or (2) when researchers want to replicate a study or perform a meta-analysis [24].

However, as Table I shows, most papers do not provide implementation details. The notable exceptions are paper [20] and paper [18], which clearly mention that their systems are built on the Apache CXF framework and the IBM WebSphere respectively. In the case of industrial case studies, sometimes the APIs are open, but the implementation techniques are kept confidential.

Summary and recommendations. The small survey that we present in this section makes it clear that comparative studies or replications are difficult to perform considering the fact that many (implementation) details are not presented in the papers considered. While this is perfectly understandable in the case of closed-source software systems, this is less so in

other cases. These observations reinforce our stance that the SOA (maintenance) community would benefit from having a standard case study in order to benchmark solutions.

When reflecting on the case studies that we came across during our small survey, we established a number of details that we would ideally want to know from all case studies:

- The implementation technology (e.g., the programming language or the communication protocol) and the used frameworks.
- The complexity of the service-based system (e.g., the number of services or interfaces).
- The availability of the system.

With these criteria in mind, we will introduce and describe Stonehenge, the standard case study system that we propose in the next section.

III. Stonehenge

Apache Stonehenge[7] is a simulation of the stock market consisting of a web application and several web services. Stonehenge provides the possibility to buy and sell shares in a single stock market, with a single currency. Apache Stonehenge was built as a joint cooperation between Microsoft and the Apache Software Foundation to showcase service interoperability between different technologies. Our goal, however, is not to explore the field of interoperability but that of maintenance in SOA, and all that it entails. We chose Stonehenge as it provides a real world example of how services can interact together to compose a software system. However, conscious of its size, we decided to extend it in order to make it more realistic and complex. We have extended it with several new features to make the system more complex on what concerns business logic and number of services. That is, we added the possibility to maintain several wallets in different currencies, to exchange money amongst the different currencies, and to use real-world data from the stock market. The result of our changes is called Spicy Stonehenge[8] which relies substantially on the business logic of the original implementation. We have also ported the original JAX-WS-based implementation to the Turmeric SOA platform[9] due to our research agenda. More on this can be found in Section IV.

A. Motivation

In our background research (see Section II) we have established that in service-oriented research there is no case study which researchers can use to compare their approaches and results. For this reason, we decided to bring forth a system that meets the criteria needed for a standard case study. For such a system we feel it is necessary that: a) it reproduces the behavior of a real-world system, b) is large or at least provides many extension possibilities that all researchers can build upon and c) it must be easy to port to different frameworks.

With Spicy Stonehenge we feel we have met these three criteria. Spicy Stonehenge provides similar behavior to that of the stock market, it is already fairly large in number of services and we plan on extending it to make it even more similar to a real system. That way, we believe Spicy Stonehenge can become the standard system which every researcher in this field can use as the "common software system" mentioned in [6].

[7]Apache Stonehenge — https://cwiki.apache.org/STONEHENGE/
[8]Spicy Stonehenge — https://github.com/SERG-Delft/spicy-stonehenge
[9]Turmeric — https://www.ebayopensource.org/index.php/Turmeric/

B. System Description

The current version of the system is composed out of five different services and two databases (Fig. 1). In this section we provide an overview of what each service does and further into the section, what data is stored in each table.

Also referring to Figure 1, solid arrows represent one service invoking another whereas the dashed arrow represent a publish/subscribe connection where the Order Processing service can subscribe to topics on the external service.

Services:

- The **Configuration Service** acts as a registry for all the deployed instances of the other services. All the other services need, therefore, to know in advance the endpoint of at least one instance of the *Configuration Service*.
- The **Business Service** mediates the interaction of the web application with the business logic of the system. For this reason, the Business Service contains all the operations the web application is capable of performing. They are: the buying and selling of stocks, user registration, statistical information about the market and information about stock prices.
- The **Order Processing Service** is solely responsible for processing the buying and selling of shares. It is meant to be invoked by the *Business Service* whenever a user performs a purchase or sale of shares in the web application.
- The **Exchange Service** makes use of Google's API for currency exchange. This service is invoked whenever the user explicitly requests for currency to be exchanged from a wallet in a certain currency into another wallet, with a different currency. In the future, this service will also become part of the purchase request for the cases when the user wants to buy shares in a currency A but chooses to use currency B.
- The **Quote Service** is in fact composed of two services. Referring to Figure 1, the service described as *Quote Service* is a normal *pull-based* service with a SOAP interface that the *Order Processing Service* can invoke to obtain data about specific stocks on-demand. On the other hand we also have the *Quote Data* service which performs two tasks: 1) it fills the *Stock Database* table with data and continuously updates it with data from Yahoo Finance, and 2) it provides a publish/subscribe interface (implemented using the ZeroMQ library) which other services, such as the *Order Processing Service* can bind to in order to be notified for price changes in specific stock symbols.

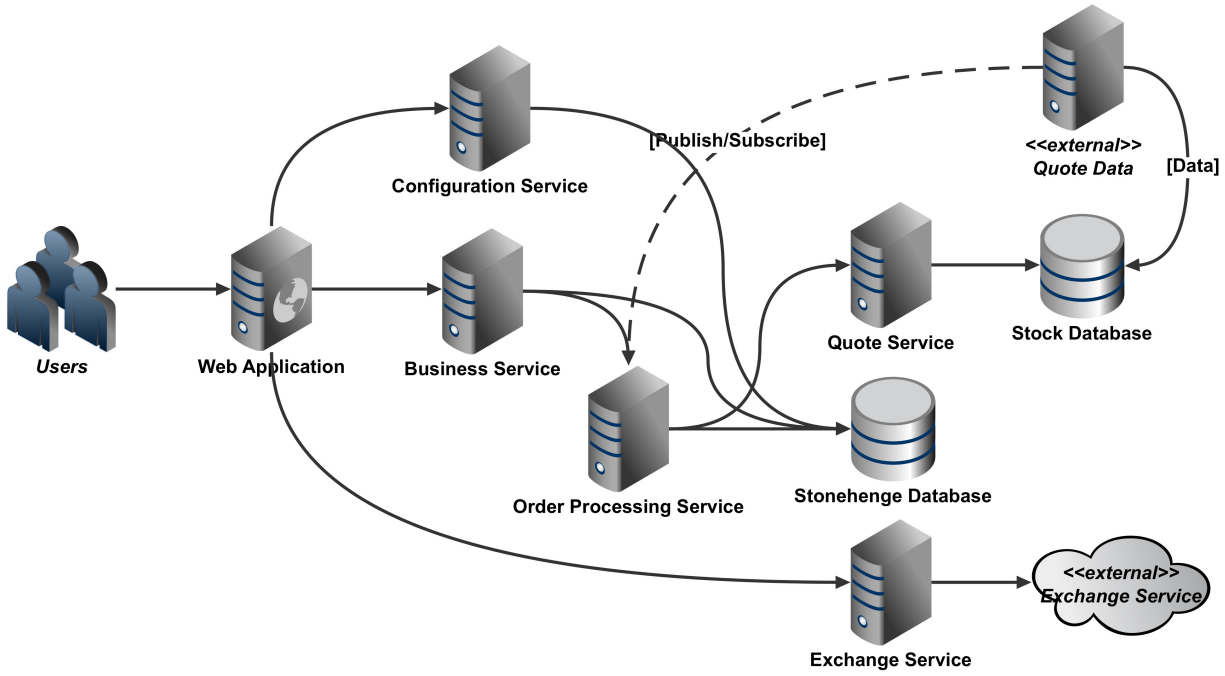

Fig. 1. Spicy Stonehenge

Databases:

- The **Stonehenge Database** contains the information necessary for the basic operation of the system. Namely it contains user information, including how much money and which stocks each user owns. It also contains information about the services' endpoints and the mapping between service instances (which instance should each service use).
- The **Stock Database** contains solely information about stock prices. This table is kept separately as it is meant to be filled by an external service which continuously checks whether there is new data and pushes it to the database.

TABLE II
FEATURES AVAILABLE/PLANNED FOR SPICY STONEHENGE

Currently available features
Purchase and sale of stocks
Price information about stock symbols
Wallets in different currencies
Management of service endpoints
User registration
Planned features
Automatic conversion of currencies
Multiple stock markets
External bank entities
Stock options

C. Usage Scenarios

With these services we can then have different usage scenarios. These are summarized in Table II.

Our planned extensions to the existing system aim at making the interactions amongst web services more complex. For example, the existence of multiple stock markets will create the need for different instances of the Order Processing service. Similarly, having an automatic conversion of currencies will add a possible additional step to the purchase of stocks.

IV. RESEARCH AGENDA

The purpose of compiling and proposing Stonehenge as a standard case study is tightly connected with our research goals. In past research we performed dynamic analysis on SOA systems in order to identify dependencies amongst services [25]. We have also investigated the challenges of performing runtime monitoring of SOA systems [26]. Our preliminary results are promising but our goal is to further investigate these topics. In order to support this earlier research, we have also ported the original Apache Stonehenge to a different platform, i.e., Turmeric SOA. This platform was built and recently open-sourced by eBay and it already provides many of the features that we require for runtime monitoring of SOA systems. This platform is also highly scalable as proven every day by supporting eBay's own business. With it we hope to replicate as closely as possible a real-world setting.

We have also been collaborating with the engineers from Intalio who are in charge of making Turmeric SOA open-source and we plan on leveraging this collaboration to learn about the pains of online SOA maintenance, reengineering and testing.

In the future we would like to investigate online evolution of SOA in more detail. This entails two major research tracks: online updating of services and versioning, that is performing an evolutionary step, and online diagnosis and testing, which consists of assessing it. Following are a few of the research questions we want to explore in the field of reengineering, evolution and maintenance:

A. Online Updating and Versioning

Performing online updates of services comes with several challenges. For instance, if the external behavior of a service changes, this might cause other services to fail (since they expect the older behavior). For this reason it is important to know exactly which services depend on which, as to have a clear picture of which services a code change can impact.

Similarly, when interfaces of services change, it might be the case that a new version of the service is created and the older one is kept for backwards compatibility. When this happens, eventually there will be a large number of versions of the same service. With runtime information about which services are no longer used we can provide software engineers with information to help mitigate this problem.

In an online system it is also important to have knowledge about the usage patterns over time. This way, maintainers can attempt to reduce downtime to a minimum by choosing the periods of time with less usage on specific services.

These concerns can be summarized as three research questions:

RQ 1.1 How can we determine which services depend on which, at runtime?

RQ 1.2 How can SOA maintainers determine the best periods to perform maintenance?

RQ 1.3 When can a version of a service be deprecated based on its usage profile?

B. Online Diagnosis and Testing

For our research, we assume that the business logic of all deployed services is tested individually, and that the integration of the deployed service in the SOA has been checked [27]. Checking the SOA during runtime then comes down to observing its behavior (through monitors) and waiting for a failure to appear. The root cause of a failure can be pinpointed through a specific lightweight diagnosis technique referred to as spectrum based fault localization (SFL) [28]. This can be used during runtime to convict or exonerate a potentially faulty service [29]. SFL is based on a matrix of service involvement in transactions carried out in the entire SOA. Based on this matrix SFL can deduce likely faulty services during runtime after a number of observations have been recorded.

However, applying SFL online creates a number of challenges:

RQ 2.1 How can we determine service involvement for SFL through monitoring and tracing of the SOA? Our goal is to determine the current architecture of the running SOA, and, therefore the width of the SFL matrix. In other words, how many services are there, and how are they interconnected at a certain point in time?

RQ 2.2 How to limit a spectrum for *online* SFL? The spectrum of a service represents which transaction it has been invoked in. Therefore, in the online case, a spectrum can become indefinitely large. This question is related to length of the SFL matrix. In other words, how much execution history is required for the diagnosis?

RQ 2.3 How can we generate test cases for observations that are required for a particular diagnosis, but that have never been made?

RQ 2.4 What is the overhead of applying online SFL to SOA? How can we determine the trade-off between the overhead of the online testing and diagnosis and the systems' performance?

Currently with Turmeric it is possible to visualize usage profiles of different services over time allowing us to have an idea of which services are most used and at what times. This feature already partly addresses **RQ 1.1** and **RQ 2.1** as it provides information about which pairs of services are invoked. One of our aims is to extend this feature and provide more detailed information about the runtime system. This way, developers in charge of maintaining SOA systems can, for example, schedule maintenance tasks appropriately and therefore lower the perceived downtime for customers.

V. Summary

The motivation in many organizations for deploying and using service oriented architectures is the loosely coupled nature of their services, facilitating flexibility in adapting systems to changing business requirements and alleviating constant system evolution. Therefore, SOAs are predestined to realize easily maintainable distributed systems.

Our research agenda addresses some of the challenges encountered when service oriented architectures are maintained while they are operating and being used by various stakeholders. We refer to this as online evolution and maintenance.

Working in this area and devising techniques for online maintenance of SOA brings with it the requirement of having a realistic case study system for experimentation, assessment, and comparison. However, when venturing into the subject, we noticed a chronic lack of suitable applications to be used as case studies. We could have used an industrial application, which would have facilitated our research goals, thereby sacrificing open access to other researchers and obstructing open discussion and exchange of ideas. On the other hand, open source SOAs are not readily available, which might well be attributable to the fact that the typical scalability requirements of organizations that lead to SOA deployment, are not apparent in the open source community.

In this paper, we have, therefore, addressed this lack by proposing an open source SOA system, i.e., Spicy Stonehenge,

which we developed out of an existing application, and now put forward as standard case study system. We hope that in the future more researchers will use and contribute to this case in order to suit their particular research interests, but also in order to facilitate comparison between all ideas and techniques developed based on this system. That way, we hope to contribute to the overall maturity of software maintenance and reengineering research in general.

In addition to developing a suitable case study for our research, we have also started to use it for our purposes, as outlined in our research agenda.

ACKNOWLEDGMENT

The authors would like to acknowledge NWO for sponsoring this research through the Jacquard ScaleItUp project. Also many thanks to our industrial partners Adyen and Exact.

REFERENCES

[1] Y. V. Natis, "Service-oriented architecture scenario," 2003, website last visited November 30th, 2011. [Online]. Available: http://www.gartner.com/DisplayDocument?id=391595

[2] N. M. Josuttis, *SOA in Practice: The Art of Distributed System Design*. O'Reilly, 2007.

[3] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding service-oriented software," *IEEE Software*, vol. 21, no. 2, pp. 71–77, 2004.

[4] B. Benatallah and H. Motahari Nezhad, "Service oriented architecture: Overview and directions," in *Advances in Software Engineering*, ser. LNCS, E. Börger and A. Cisternino, Eds. Springer, 2008, vol. 5316, pp. 116–130.

[5] S. Benbernou, L. C. M. S. Hacid, R. Kazhamiakin, G. Kecskemeti, J.-L. Poizat, F. Silvestri, M. Uhlig, and B. Wetzstein, "State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs," 2008, deliverable # PO-JRA-1.2.1 of the S-Cube project.

[6] S. E. Sim, S. M. Easterbrook, and R. C. Holt, "Using benchmarking to advance research: A challenge to software engineering," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2003, pp. 74–83.

[7] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé, "Applying and combining three different aspect mining techniques," *Software Quality Journal*, vol. 14, no. 3, pp. 209–231, 2006.

[8] S. Demeyer, T. Mens, and M. Wermelinger, "Towards a software evolution benchmark," in *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE)*. ACM, 2002, pp. 172–175.

[9] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time monitoring of instances and classes of web service compositions," in *Proceedings of the International Conference on Web Services (ICWS)*. IEEE Computer Society, 2006, pp. 63–71.

[10] M. Pistore and P. Traverso, "Assumption-based composition and monitoring of web services," in *Test and Analysis of Web Services*, L. Baresi and E. Di Nitto, Eds. Springer, 2007, pp. 307–335.

[11] B. Domenico and G. Carlo, "Monitoring conversational web services," in *Proceedings of the 2nd international workshop on Service oriented software engineering (IW-SOSWE)*. ACM, 2007, pp. 15–21.

[12] G. Heward, I. Müller, J. Han, J.-G. Schneider, and S. Versteeg, "Assessing the performance impact of service monitoring," in *Australian Software Engineering Conference (ASWEC)*. IEEE Computer Society, 2010, pp. 192–201.

[13] A. Marconi and M. Pistore, "Synthesis and composition of web services," in *Formal Methods for Web Services*, ser. LNCS, M. Bernardo, L. Padovani, and G. Zavattaro, Eds. Springer, 2009, vol. 5569, pp. 89–157.

[14] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, "Fault tolerant web service orchestration by means of diagnosis," in *Proceedings of the Third European Workshop on Software Architecture (EWSA)*, ser. LNCS, vol. 4344. Springer, 2006, pp. 2–16.

[15] L. Baresi, C. Ghezzi, and S. Guinea, "Smart monitors for composed services," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC)*. ACM, 2004, pp. 193–202.

[16] K. Mahbub and G. Spanoudakis, "Run-time monitoring of requirements for systems composed of web-services: Initial implementation and evaluation experience," *Proceedings of the International Conference on Web Services (ICWS)*, pp. 257–265, 2005.

[17] C. Momm, R. Malec, and S. Abeck, "Towards a model-driven development of monitored processes," *Internationale Tagung Wirtschaftsinformatik (WI2007), Karlsruhe*, 2007.

[18] K. A. Nasr, H.-G. Gross, and A. van Deursen, "Realizing Service Migration in Industry - Lessons Learned," *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, 2011.

[19] A. Ahmad and C. Pahl, "Customisable transformation-driven evolution for service architectures," in *Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society, 2011, pp. 373–376.

[20] B. R. Schmerl, D. Garlan, V. Dwivedi, M. W. Bigrigg, and K. M. Carley, "Sorascs: a case study in soa-based platform design for socio-cultural analysis," in *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 2011, pp. 643–652.

[21] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli, "Automatic synthesis of behavior protocols for composable web-services," in *Proceedings of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2009, pp. 141–150.

[22] G. Denaro, M. Pezzè, and D. Tosi, "Ensuring interoperable service-oriented systems through engineered self-healing," in *Proceedings of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2009, pp. 253–262.

[23] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Kluwer, 2000.

[24] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[25] T. Espinha, A. Zaidman, and H.-G. Gross, "Understanding service-oriented systems using dynamic analysis," in *Proceedings of the International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*. IEEE Computer Society, 2011, pp. 1–5.

[26] C. Chen, A. Zaidman, and H.-G. Gross, "A framework-based runtime monitoring approach for service-oriented software systems," in *Proceedings of the International Workshop on Quality Assurance for Service-Based Applications (QASBA)*. ACM, 2011, pp. 17–20.

[27] A. Bertolino and A. Polini, "Soa test governance: Enabling service integration testing across organization and technology borders," in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*. IEEE Computer Society, 2009, pp. 277–286.

[28] É. Piel, A. González-Sanchez, H.-G. Groß, and A. J. C. van Gemund, "Spectrum-based health monitoring for self-adaptive systems," in *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2011, pp. 99–108.

[29] T. W. Reps, T. Ball, M. Das, and J. R. Larus, "The use of program profiling for software maintenance with applications to the year 2000 problem," in *Proceedings of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 1997, pp. 432–449.

TUD-SERG-2012-001
ISSN 1872-5392