

Understanding the Interactions between Users and Versions in Multi-Tenant Systems

Tiago Espinha
Delft University of Technology
Delft, The Netherlands
t.a.espinha@tudelft.nl

Andy Zaidman
Delft University of Technology
Delft, The Netherlands
a.e.zaidman@tudelft.nl

Hans-Gerhard Gross
Delft University of Technology
Delft, The Netherlands
h.g.gross@tudelft.nl

ABSTRACT

Multi-tenant systems represent a class of software-as-a-service (SaaS) applications in which several groups of users, i.e. the tenants, share the same resources. This resource sharing results in multiple business organizations using the same base application, yet, requiring specific adaptations or extensions for their specific business models. Each configuration must be tended to during evolution of a multi-tenant system, because the existing application is mended, or because new tenants request additional features. In order to facilitate the understanding of multi-tenant systems, we propose to use a runtime topology augmented with user and version information, to help understand usage patterns exhibited by tenants of the different components in the system.

We introduce *Serviz*, our implementation of the augmented runtime topology, and evaluate it through a field user study to see to which extent *Serviz* aids in the analysis and understanding of a multi-tenant system.

Categories and Subject Descriptors

D.2 [Software]: Software Engineering; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

General Terms

Experimentation

Keywords

Web service maintenance, runtime topology, multi-tenancy, user-study

1. INTRODUCTION

Multi-tenant systems represent a class of software-as-a-service (SaaS) applications in which several groups of users, i.e. the tenants, share the same (software) resources [1]. A

multi-tenant system allows to make full use of the economy of scale, as several SaaS customers access the same application at the same time, while the instances are configured according to the diverse requirements of the various tenants [15]. Multi-tenant systems are characterized by high demands on configurability and evolvability, which go well beyond multi-instance and multi-user applications, or software product lines [3]. These demands are instigated by the multiple tenants, or business organizations, that all use the same base application, yet, require specific adaptations or extensions in order to suit their specific business models.

Since flexibility is key to effective multi-tenant applications, their deployment should go along with inherently accommodative architectures, such as service oriented architectures (SOA) [21]. For example, SOA permit to isolate tenant-specific business needs in a separate version of a service, which raises the challenge of managing versions of services. This is evidenced by the work of Fang et al. [10] who describe an approach to manage different service versions in a web service directory.

It has long been recognized, that software systems must evolve in order to remain successful [16]. This is no different in multi-tenant software systems, and we hypothesize that evolution might even be more important. This is due to the large number of stakeholders involved, each demanding new features to be added to (their version of) the multi-tenant software system. This brings about the challenge of having to understand the impact of changes on different versions of the code (deployed as different versions of services) that might be specific to a single tenant or a group of tenants.

Even though SOA provides the flexibility for realizing multi-tenant software systems, we know from Gold et al. that the promise of easier software maintenance is not completely met [11], because even though conducting the actual evolution in terms of exchanging services is possibly easier, the understanding of their interactions and usage is still an issue [11]. This comes from understanding a monolithic application versus understanding a distributed system composed of many entities (services).

With the increased complexity of multi-tenant software systems [1], their understanding is likely to become more demanding. Corbi claims that up to 60% of the maintenance effort lies in understanding the system [4]. Strangely enough then, a large-scale survey of scientific literature in the area of program comprehension using dynamic analysis from 2009 did not reveal any major advances in the area of understanding SOA based systems using dynamic analysis. “This seems strange, as dynamically orchestrated com-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IWPSE '13, August 19-20, 2013, Saint Petersburg, Russia
Copyright 13 ACM 978-1-4503-2311-6/13/08 ...\$xx.00.

positions of services would benefit from dynamic analysis for understanding them [5]”, because the composition of the entire SOA-based system only happens at runtime and its configuration can change during operation [2, 23], leading to highly dynamic systems. It comes as no surprise then, that Kajko-Mattsson et al. identified the understanding of the SOA infrastructure as one of the most important challenges when evolving SOA-based software systems [14].

In previous work [9] we presented the runtime topology as a means to aid the understanding and maintenance of service based systems. Our preliminary results show that the runtime topology is indeed a good means to understand highly dynamic SOA-based software systems. In this paper we extend the runtime topology to also incorporate user and version information, since we believe that these help to understand the role of multi-tenancy in SOA-based software systems. They show how different tenants use the system, and how several versions of similar services are customized for particular business needs in different ways.

This leads us to our **main research question**: *Does the combination of user, service-version and timing information projected on a runtime topology help in the understanding of SOA-based multi-tenant software systems?*

In order to steer our research, we will investigate these subsidiary research questions:

RQ1 Does the user information added to the runtime topology help in the understanding of SOA-based systems?

RQ2 Does the service version information added to the runtime topology help in the understanding of SOA-based systems?

RQ3 Do usage graphs help to understand SOA-based systems?

In order to evaluate the approach, we set up a user study that involves our research prototype *Serviz* and a case study system called Spicy Stonehenge [8], a stock-market simulator. The user study is set up as a contextual interview and it involves four software engineering professionals.

The remainder of this paper is structured as follows: Sections 2 and 3 describe our approach (the data requirements and how we collect the data) and the tool implementing it. Section 4 outlines the experimental setup, and Section 5 presents its results. Section 6 discusses the results and potential threats to validity. The paper is rounded off with related work and conclusions in Sections 7 and 8.

2. APPROACH

Central to our understanding approach stands the runtime topology of a service-oriented software system, representing the configuration of a set of services that are deployed within an environment. Based on previous research [9], we found to be essential to include information which allows us to link service invocations to request traces and associate each request with its timestamp. This information alone allows us to infer the runtime topology of a running service-based system. In addition, and due to our focus on multi-tenant systems, usage scenarios require different versions of the same service to co-exist in the same platform. These versions can then be used by different tenants who, regardless of the client used to invoke the service, may cause different usage patterns.

In order to satisfy this requirement, we also included information regarding which user caused a particular invocation as well as which version of a service was invoked. In the following subsection we present the data requirements inherent to creating the initial runtime topology of a service-oriented system, plus the additional data required in order to be able to add the user and service version dimensions.

2.1 Data requirements

The runtime topology on its own has specific data requirements which we already identified earlier, and whose extraction in our platform of choice (Turmeric SOA¹) is briefly described below in subsection 2.2. These requirements represent the essential set of data required for plotting a runtime topology diagram representing the service invocations. The collected data are:

- **Request ID** - A unique identifier per invocation trace. It allows us to link pairs of services as belonging to the same trace.
- **Timestamp** - The current timestamp at the time of the request, as seen by the database server. This is crucial for analyzing exactly when the web services were busiest.
- **Consumer name and method** - The name of the caller service (i.e. client) and the respective method which caused this request pair.
- **Service name and method** - The name of the callee service in this request pair and the respective web method being called.

More in-depth detail on the reasoning behind the usage of this data can be found in our previous work [9].

In our current research, we analyze the usefulness of the runtime topology for multi-tenant systems. This type of systems, due to its inherent usage patterns (i.e. multiple users simultaneously use multiple service versions), adds two important dimensions which need to be taken into consideration upon performing maintenance. For this reason, our approach requires additional data to be added per service request. Therefore, our approach now also includes:

- **Consumer and service versions** - For each pair of calls, the consumer and service versions are stored.
- **Username** - The username of the user who originally caused the request to happen.

Having identified all the data requirements for our particular goal, in the next section we propose an approach to collect the data from a running system.

2.2 Data extraction

The data extraction step required to enable the runtime topology is highly dependent on the platform used. Some service platforms may already provide parts of the required data whereas others may require deeper changes in order to obtain such data. Ultimately, however, this is a completely automated step after it has been enabled for a specific platform. This means the data is automatically pulled from the running system rather than requiring manual intervention.

¹Turmeric SOA — <https://www.ebayopensource.org/index.php/Turmeric/>



Figure 1: Screenshot of Serviz

For our implementation we chose the Turmeric SOA platform, the open source version of eBay’s services platform, as it already provides some of the information we described in the previous subsection. This data is not provided by default in Turmeric SOA and in order to collect it, we make use of one of the framework’s features. This allows us to intercept each incoming request by means of a Java class which can then access and manipulate information regarding the request. In order to retrieve information on invocation pairs, we had to follow another solution: we appended information to the request string, thus making sure that information on the caller got to the callee. Since this data is only available during the web service’s invocation lifetime, the data must be stored persistently. For this reason, the request handler stores the data in a MongoDB database in order to keep a historical track of the system’s usage.

In practice, we are able to collect all the required data by handling the incoming requests from the point of view of each web service. This event provides us with all the information mentioned in Subsection 2.1.

3. SERVIZ

Serviz² is an open-source visualization tool which allows software engineers to visualize periods of high and low usage in a service-based system. Additionally, Serviz allows this information to be filtered for a set of users, per service and version, and according to a specific time-frame.

In previous work [9] we performed a pre-test post-test experiment on whether the use of a runtime topology of a service-based system would help in understanding such a system. From this experiment we gathered that while the runtime topology alone provides a step forward in understanding service-based systems, it can be enhanced by pro-

viding more runtime information to system maintainers.

With this in mind, we enhanced Serviz with new features. They are: the capability to filter the information displayed based on which users were using the system at the time, and similarly, the capability to filter such information based on a specific service name and version. Simultaneously, we added histograms to more easily visualize service usage, per service, over time. Having these added filtering capabilities provides the maintainers with more detailed insight into how the system has behaved and is behaving. A screenshot of Serviz can be seen in Figure 1. On the left hand side of the screenshot, we see the filtering options for the collected data: the time-period under consideration, the users and also the versions of interest. The right hand side of the screenshot shows the different services and the service methods called per service (below the service name and invocation frequency). Of interest here is that the `OrderServiceProcessor` has two versions, which each play an active role in the functioning of this service-based system.

We now discuss some of the main features of Serviz.

User filtering

It is important for a service provider managing a service-based system to be able to filter the runtime usage based on specific users. For instance, after a system maintainer has inferred that a service is only used by a specific user (e.g. in the case when a specific version was created to cater the specific needs of a particular user), the maintainer can then pinpoint high and low peak usages for that specific user by filtering this data.

Service/Version filtering

By filtering the runtime topology with the service and version axes, maintainers are able to find services which are direct dependences to a particular service version. In the

²<https://github.com/etiago/serviz>

same way, by selecting extended periods of time, maintainers are also able to determine when a particular version of service can be exchanged for a newer version with minimal interruption.

Combined filtering

Our runtime topology also allows system maintainers to perform combined filtering using both the user and version filtering axes. This way maintainers can find out information such as “which service versions have never been used by a particular user” or through knowing that a particular version is only used by a specific user, maintainers can find out periods suitable for maintenance.

Histograms

Another feature of Serviz are the time-based histograms with service usage over time (see Figure 2). The histograms are created per service and also take into consideration the filtering defined by the maintainer. The histograms are to be combined with the filtering features, so that maintainers can use them not only to determine periods of high and low usage for the whole system, but also on a per user and per service basis. This allows for a very fine grained view of which users use which services the most and at which times.

4. EXPERIMENTAL SETUP

In order to evaluate our implementation of a runtime topology augmented with time, user and version information, we performed a field user study with a total of four software engineers from two IT companies in the Netherlands, Adyen B.V. and Exact N.V. The user study was done in two sessions, one session at each company,

This augmented runtime topology is especially aimed at multi-tenant systems, where different users use different parts of the system in different periods of time. Because of this added complexity, we involved software engineers from IT companies which are dealing with multi-tenant scenarios as part of their own software systems. This user study was organized as a contextual interview [13, 18, 25].

Step 1: Demo of Serviz

We started the session with a short demo of Serviz in which we showed the developers all features of Serviz.

Step 2: Free exploration

In this step we asked the developers to freely explore Spicy Stonehenge, which we had pre-installed and ready to be used with Serviz. We gave them the goal of getting a good under-

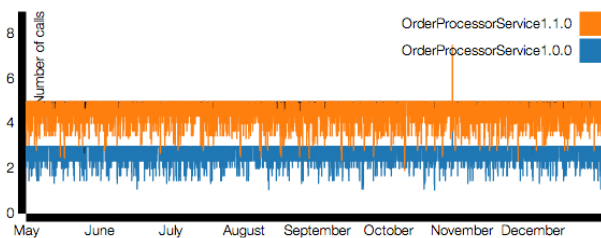


Figure 2: Histogram showing the usage (requests per minute) of two versions of the same service.

standing of the implementation of the major functionality in Stonehenge *and* how the system is used based on the accompanying usage data. We told them that we would discuss the implementation details of this functionality later on during the session.

This phase of free exploration took in both cases less than one hour and was done in pairs, as to stimulate communication between the participants during exploration, thereby simulating the think-aloud protocol [7]. We recorded the conversation between the software engineers and provided assistance whenever the information shown was not clear enough. As an example of this, Serviz was loaded with data solely for the year 2012, so to prevent the participants from losing time, this information was provided whenever the participants tried to input dates in 2013. Similarly, whenever the directional arrows linking the service were not clear enough, this information was provided to the participants.

Step 3: Questionnaire

When the developers were satisfied with their reconnaissance of Spicy Stonehenge, we presented them our questionnaire (shown in Table 1) which had to be answered individually. This questionnaire is not so much meant to gather quantitative data, but rather serves as a means to steer the discussion in the next step, which is aimed at gathering qualitative data.

Step 4: Contextual interview

While we already gained quite a lot of information during the free exploration phase, we intensified the interview once the developers felt they were comfortable with Serviz and the case study system Spicy Stonehenge. In particular, we used a contextual interview [13]. This contextual interview already started during the second step, where we observed how the developers explored the system. In particular, we took note of which questions they were asking and how they were using Serviz to answer these questions. Subsequently, we continued the interview and we aimed to further explore the possibilities of Serviz and identify circumstances in which Serviz can be of benefit. In order to steer this conversation, we used the questionnaire from Table 1 as a basis.

Again, this discussion was recorded. The interview took on average one hour and a half and a time limit was not imposed on the participants. This excludes any possibility for time-related pressure to finish the experiment.

4.1 Case study system

The subject system that we pre-loaded into Serviz is Spicy Stonehenge, a simulation of the stock market [8]³. Spicy Stonehenge is composed of 6 services and the user data that we preloaded concerns 3 users. An overview of Stonehenge can be seen in Figure 3. The figure depicts two services with multiple versions (BusinessService and ConfigurationService), which are in fact fully independent implementations of each service. This particular setup was chosen due to the resemblance it bears to multi-tenant systems. In such systems, user-specific configuration is created by branching versions of services containing the business logic required by a specific user.

Important to note is that none of the participants was

³<https://github.com/etiago/spicy-stonehenge>

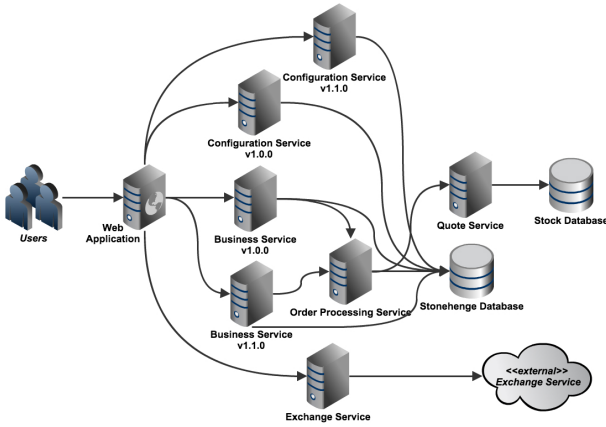


Figure 3: Spicy Stonehenge

familiar with (Spicy) Stonehenge.

4.2 Questionnaire

The origin of the questionnaire can be traced back to the work of Sillito et al. [19], in which the authors present a set of 44 typical questions that software engineers have when performing maintenance on a software system. However, a number of questions from this work was either not relevant for the context of service-based systems or had to be adapted to this context. For example, the original question “What is the behavior that these types provide together and how is it distributed over the types?” is now rephrased as “What is the behavior these services provide together and how is it distributed over the services?” (SE11). Despite these changes, we tried to preserve the main phrasing of these questions as much as possible. The questions should then be interpreted as whether *Serviz* helps in achieving each of those goals (i.e. finding a particular behavior or pattern). From the original 44 questions from Sillito et al. we kept 19 questions that are particularly relevant for *Serviz*, e.g., questions that dealt with static analysis were removed. Additionally, we introduced 20 new questions, which we feel are important questions when trying to understand a service-based system. All questions are listed in Table 1 and they are organized as follows: G1 through G4 are general questions about the usability and usefulness of *Serviz*, SE1 through SE19 are questions extracted from Sillito’s work, U1 through U4 are questions related to the user filtering feature, S1 through S4 concern the service filtering, V1 through V4 relate to the version filtering and lastly, C1 through C3 identify the user experience with combined filtering of the previous features.

The participants were asked to fill in the questionnaire using a 5-point Likert scale ranging from *strongly disagree* to *strongly agree*. At the end of the questionnaire there were 2 open questions gauging for the most-liked features of *Serviz* and asking for any additional comments or suggestions.

4.3 Participants

For this study we approached four software engineers from two distinct software companies. All four participants have extensive experience with web service development (> 2 years). However, the technologies used and the context in which they are used are quite different. This is reflected in the results of the experiment where clear disagreements

exist and are explored later in the paper.

Adyen, a SMB, makes extensive use of web services in their business practice and the majority of the technologies used are developed in-house. This leads to a deep knowledge of every single component of their own system. It also means that every developer has a very good understanding of how the system interacts to bring together functionality.

As for Exact, while this is by comparison a larger company with around 3000 employees, service orientation is being actively pursued. There is also a larger reliance on proprietary third-party technologies which can sometimes obscure the runtime details of a software system.

5. RESULTS

The quantitative results obtained through the four participants of the questionnaire are presented in Table 1. The subsequent interview was done in pairs of two participants and discussed each of the questions in the questionnaire. Our discussion below will highlight those questions that sparked the most interesting discussions.

An important note concerning the results table is that the results P1 and P2 represent the participants from Adyen whereas P3 and P4 represent those from Exact.

5.1 General questions

The general questions assess the usability of *Serviz*, whether it would save the maintainers time while maintaining the system, and also whether the runtime topology helps with understanding service-oriented systems during maintenance.

These first results provide a somewhat mixed image of how useful *Serviz* is perceived to be. There is a clear divide amongst the companies, with the participants from Adyen generally giving lower marks. In particular, when asked whether *Serviz* would save time and whether it would help them understand a service oriented system during maintenance, they seemed more reluctant by scoring these questions with a 2 (*disagree*). The two participants from Exact, on the other hand, gave a score of respectively 4 (*agree*) and 5 (*strongly agree*). This divide can be explained by the context in which both companies are operating, something which is also supported by the participants’ comments during the interview. More specifically, P1 claimed “I think this is useful to identify the topology of your system, but what happens if you already know the topology, in the beginning you want to get to know the system then it can be useful”. In this case, participants P1 and P2 claimed during the interview that because they are well acquainted with their system’s topology, they did not see the usefulness of *Serviz*.

However, they recognized the value for immigrants to the software system by stating “I liked it as a tool you can use at the beginning”. This is in line with the observations of Sim et al. [20] who state that “software immigrants” need to undergo a naturalization process during which they get to know the system. This notion of “software immigrants” deserves particular attention in the case of large multi-tenant software systems. Here, interactions are determined at runtime and the services interact in various ways to satisfy the requirements of different users. Even well-seasoned software engineers who are well acquainted with the system may feel as if they were *immigrants* to a particular usage workflow of a particular user.

Additionally, from the interview we identified another reason why for participants P1 and P2 *Serviz* may have limited

		Adyen Exact			
		P1	P2	P3	P4
G1	The tool was easy to use	3	3	4	3
G2	A tool like Serviz will save me time	2	2	4	4
G3	Visualizing the runtime topology makes maintenance tasks on service oriented systems easier	3	5	5	5
G4	A tool like Serviz will help me understand and maintain a service oriented system	2	2	5	4
SE1	Which service represents this UI element or action.	3	3	4	4
SE2	Where is there any code involved in the implementation of a particular behavior	4	4	3	3
SE3	Where is there an exemplar for a certain behavior?	3	4	5	4
SE4	Identifying system services based on their names.	4	4	5	4
SE5	What are the interfaces of a specific service?	4	2	4	4
SE6	Where is a method called?	4	2	4	2
SE7	When during the execution is a method called?	2	1	2	2
SE8	Where are instances of a service created?	3	1	4	2
SE9	How are services composed/assembled to bring together functionality?	3	3	5	4
SE10	How is a feature implemented?	2	2	4	3
SE11	What is the behavior these services provide together and how is it distributed over the services?	2	1	4	4
SE12	How is control getting from here to there?	2	2	4	3
SE13	Why isn't control reaching this point in code?	2	2	3	2
SE14	Which execution path is being taken in this case?	4	1	4	3
SE15	How does the system behavior vary over these services?	2	1	4	3
SE16	What is the mapping between these services?	4	2	5	4
SE17	To move a certain feature, what else needs to be moved?	2	4	5	2
SE18	What will be the direct impact of a change?	3	4	5	3
SE19	What will be the total impact of a change?	3	4	4	3
U1	It was clear I could filter the runtime topology based on usernames.	4	5	5	4
U2	The username filtering adds value to the runtime topology.	5	5	5	4
U3	By using the username filtering, I could find out which users use the system the most in a specific time interval.	2	3	4	4
U4	By using the username filtering, I could find out which users have used the system the most.	2	3	4	4
S1	It was clear I could filter the runtime topology based on specific services.	1	4	4	4
S2	The service filtering adds value to the runtime topology.	1	3	5	4
S3	By using the service filtering, I could find out which services are used the most.	1	3	2	3
S4	Service filtering helps me find out which services depend on each other.	1	3	4	3
V1	It was clear I could filter the runtime topology by picking specific versions of services.	4	4	5	4
V2	The version filtering adds value to the runtime topology.	3	4	5	4
V3	By using the version filtering, I could find out which services are good candidates for dead code (never used).	1	4	4	4
V4	Version filtering will help me pinpoint which service version to perform maintenance on.	1	4	5	3
C1	With version and user filtering I could find which user has used a specific version the most.	2	3	4	4
C2	Version and user filtering allows me to find which versions a user never used.	4	3	3	4
C3	With version and user filtering I can find periods of low usage which are more suitable for software maintenance.	2	4	5	4

Table 1: Questionnaire

usefulness. As the participants stated, at Adyen, the user-specific variation is not done through service versioning. Instead, their system resorts to inline configurability in the source code. Because Serviz acts at the service level, it does not provide source code granularity, and it might be difficult for participants P1 and P2 to imagine how Serviz would address their scenario.

As for the Exact case, both participants P3 and P4 agreed that the tool was easy to use (4 points and 3 points), that it will save them time (4 points for both), that it makes maintenance tasks easier (5 points for both) and that it will help them understand and maintain a service oriented system (5 points and 4 points). When asked how it helps them during their understanding and/or maintenance, P3 and P4 mentioned that they do not have any tooling available which shows them the information that Serviz provides.

In contrast, in the post-questionnaire discussion, Adyen's participants revealed a prototype tool developed in-house which bears similarities to the histogram feature of Serviz. This tool was driven by their own internal needs and in the specific case of Adyen, the histogram works in real-time. Simultaneously, it also lists the users which have been using the system recently, ranked by number of requests. This demonstrates that even in a slightly different setup like Adyen's, the need exists to know how the users of a service-based system are actually using the system.

5.2 Generic software engineering questions

In this section we analyze the results regarding the general software maintenance questions that are based on the questions found by Sillito et al. [19]. Here the answers are also mixed and require deeper analysis.

The first major disparity in the scores begins with question SE5 (What are the interfaces of a specific service?), where participants P1 and P2 scored the question with 4 and 2 points respectively. These results are intriguing at a first sight, but through inquiring the participants about these scores, we realized participant P2 had different expectations from the "service interface" nomenclature. Namely, participant P2 did not consider the method names as the interface of a service as just the method name does not include, for example, the method signature.

Question SE7 has generally low scores with all participants scoring it with 2 points except for P2 which scored 1 point. Upon further investigation with the participants, the general opinion was that more detail was expected regarding the "when during the execution is a method called". In fact, the participants claimed they were expecting to know the exact line of code to be able to infer the "when" rather than just knowing the method in which another method is called. This appears to simply have been a mismatch between expectation and reality.

Question SE8 presents a similar scenario with participants P2 and P4 expecting more detail about the *where* the instances of a service are created. Another remark was the existence of a misunderstanding. Participant P2 said he did not know where the service instance was created, as with some implementations this happens within the service framework. Here again, a misunderstanding happened with the definition of the "creation of a service". However, the participant did agree that it was possible to identify the place where the preparation of a service call is created.

Questions SE9, SE10 and SE11 focus on service composi-

tion and how the services come together to provide functionality. In all of these questions the major divide is between the two different companies, with Adyen's participants scoring these questions much lower than Exact's counterparts. This can again be accounted for with the nature of the systems the participants are used to. Because P1 and P2 (from Adyen) deal with much more stable software systems, it can be more difficult to see the added value of using runtime information to figure out how a feature is implemented (SE10), or how a behavior is distributed over services (SE11).

On the other hand, because Exact has a larger software system with more variation, it seems easier for the participants to understand the value of finding out exactly how a certain feature is implemented in terms of services, or how a behavior is distributed across the different services. Since service-orientation is also something not yet fully in practice at Exact, it might also be the case that the participants do not have as much bias from the systems they are used to.

5.3 User filtering

When asked whether it was clear that it was possible to filter the runtime topology based on usernames (U1), the results were on the overall very positive (scores of 4 or more). Also when asked whether this feature adds value to the runtime topology (U2), the participants agreed, with only one participant giving 4 points and all the rest scoring it 5 points.

Some disagreement seems to exist on whether this feature helps in finding out which users use the system the most during a specific time interval (U3). Participants P1 and P2 scored it with 2 and 3 points respectively, whereas participants P3 and P4 scored this question with 4 points. The same question was asked, but then with a broader scope regarding time rather than focusing on a specific interval (U4). The results for this question were identical.

The lower scores from participants P1 and P2 can be explained also through weighing in the interview transcripts. In the interview, the participants claimed they would have liked the possibility to have this done automatically rather than having to find the users manually. Namely, the developers considered the task of having to manually input each username one by one to be cumbersome and time-consuming. This is a valuable insight, which we intend to implement in a future release of Serviz.

5.4 Service filtering

For service filtering, there was a disagreement within Adyen's participant group. All the questions received a 1 point mark from P1 where as the remaining participants are generally more positive about the service filtering features. Namely when asked about how clear the feature's availability was (S1), all participants scored it with 4 points. As to whether it adds value (S2), the results are all positive in the overall with the participants P2, P3 and P4 ranking it 3 points, 5 points and 4 points respectively.

This feature was also combined with the histograms which the participants seemed to have had trouble with. Two participants (P1 and P3) ranked this feature with 1 and 2 points respectively, whereas P2 and P4 ranked this feature with 3 points. From the interview it transpired that this was mainly caused by the runtime data not containing actual cases of a service being used more or less than another.

As for the question on whether the service filtering feature helps in finding out which services depend on each other,

participants P2, P3 and P4 ranked it with 3, 4 and 3 points respectively, which is a reasonably positive score.

The fact that P1 answered all the questions regarding service filtering with 1 point can also be attributed to the fact that P1 did not identify any usefulness for Adyen's particular environment. This is also somewhat reflected in P2's answers, which despite more positive than P1's, are also generally lower scored than those of P3 and P4.

We conclude that the ability to filter the runtime topology based on particular services is mostly useful for systems with: a) a large amount of services and b) little insight about service usage. In the particular case of Adyen where the participants already possessed a very good understanding of the software system, it would seem that such a feature would provide no added value. Furthermore, Adyen's system also does not face regular changes in topology. It is also our understanding that for this particular system, there is a great deal of logging being done down at the line level. While this provides tremendous insight for post-mortem analysis when failures occur, it also generates a great amount of data which needs to be stored.

5.5 Version filtering

On what concerns version filtering, the results were on the overall positive. Despite being clear to all participants that this filtering was available (V1), disagreements exist regarding whether it adds value (V2) and the overall usefulness of Serviz for different version-related tasks (V3 and V4). Namely, participant P1 ranked once again the usefulness questions with 1 point. This was something we pursued in depth during the interview that came after the questionnaire and it came to light that Adyen's system does not possess explicit versioning. This eliminates the needs to filter usage per version, as the per-user configurations are done through conditional code calls.

Examining the question on dead code inference based on the runtime topology filtered by service version (V3), all but one participant agreed with a 4 point score. This feature is something which during the experiment left much to the imagination. Because the data under analysis did not actually include any event of a service becoming dead code, this was something the participants had to imagine rather than something they could in fact see during the experiment. This could explain why P1 scored the feature with a lower score compared to the remaining participants.

Lastly, as to whether it helps pinpoint which services to perform maintenance on (V4), the results are mixed. Participants agreed that this is something which could be possible should there be a bug which completely disables a service call. The only remark which came to light regarding this question came from P1 who expected an automated failure detection. Currently, the approach involves comparing periods of time when the system is healthy, with periods of time when a problem occurred. Then, with the information at hand, try to find out where the problem might exist. Automated failure detection is not something we tried to achieve with our current research. For this particular question, the remaining participants (P2, P3 and P4) ranked it with respectively 4, 5 and 3 points, therefore agreeing that the version filtering feature helps figuring out which service version requires maintenance.

5.6 Combined filtering

In the previous sections we analyze the results for the individual filtering options used on their own. In the last round of questions we analyze to what extent all the features combined help in different maintenance tasks. Namely, we asked the participants whether it was possible, using version and user filtering, to identify which user accesses a particular version the most (C1). The results continue the trend of previous questions with participants P1 and P2 scoring lower (2 and 3 points respectively) and participants P3 and P4 providing more positive results (4 points for both participants). The lack of explicit versions in Adyen’s software system makes the participants less prone to find the usefulness of version-related features. Additionally, participants P1 and P2 had claimed previously about the user-specific questions that it is a cumbersome task to manually filter through all the users. This is a possible explanation to the lower scores of these participants compared to the 4 points attributed by participants P3 and P4.

Regarding question C2, because of its focus on a particular user, the results are generally higher across all the participants and there is no major discrepancy between Adyen’s and Exact’s participants (4 points, 3 points, 3 points and 4 points).

6. DISCUSSION

In this section, we discuss the results of the experiment with regards to the research questions. Subsequently, we talk about our lessons learned and we identify threats to validity.

6.1 The research questions revisited

RQ1: *Does the user information added to the runtime topology help in the understanding of SOA-based systems?* We asked the four participants of our user study for their opinion on the usefulness of the runtime topology for understanding SOA-based systems. They all expressed themselves positively here, giving a first clear indication that user information does indeed help to understand the system and how it is used. The participants’ appreciation for this feature is likely related to the multi-tenant requirement of knowing which tenants are using which parts of the system.

RQ2: *Does the service version information added to the runtime topology help in the understanding of SOA-based systems?* The opinions of the four participants somewhat diverged from each other on this point. In particular, participant P1 is of the opinion that the runtime topology is mainly useful for software immigrants, and not for software engineers who already have a thorough knowledge of the system, especially for systems that do not change very often and/or do not have explicit service versioning.

However, three participants do agree that in systems that change often and that do have explicit service versioning, the ability to add service versioning information to the runtime topology — and also filter on versioning information — helps in trying to understand a software system.

The same three participants were also positive about the fact that the service filtering feature helps them to better understand which services depend on each other.

RQ3: *Do usage graphs help to understand SOA-based systems?* Most participants think that the usage graphs available in Serviz provide a good combination with the user filtering, this way they could get a clear view of how a (set

of) user(s) uses the system throughout time.

It should also be noted that with regard to the usage graphs, this is a feature of which an experimental prototype is being independently developed at one of the companies. This on itself highlights that this particular feature represents a real-world need.

Having discussed the subsidiary research questions, we are now in a position to answer our main research question: *Does the combination of user, service-version and timing information projected on a runtime topology help in the understanding of SOA-based multi-tenant software systems?* The results give a clear indication that most of the participants to our field study are positive towards Serviz and its features to better understand how users and service-versions interact with each other over time. We also specifically gauged for whether it is possible to find periods of low usage (for particular service-versions) which would prove more suitable for performing software maintenance when version *and* version filtering is combined. The results (2, 4, 5 and 4 points) indicate that this particular filtering combination helps in finding periods of low usage.

6.2 Lessons learned

The field study with 4 participants at two different companies also taught us a number of valuable insights that go beyond the augmented runtime topology.

In particular, for a number of questions we observed quite different answers from the participants belonging to the different companies. While all four participants are experts in the area of SOA-based software systems, the particular *context* of the company plays an important role in their appreciation. This underlines the importance of performing program comprehension experiments with participants that are from different contexts.

We witnessed a generally lower appreciation of Serviz by participant P1. From the interviews that we held as the final step of the field study, we gathered from him that he thought that Serviz was not particularly useful if you already have a good understanding of the system. Yet, he also acknowledges that novice users, the so-called software immigrants, would actually benefit from the overview that Serviz could give them. In effect, this seems somewhat strange, because at the beginning of the field study, we explicitly asked them to reason about Serviz from the point of view of an engineer exploring an unknown system (hence also the choice for Stonehenge, our subject system, which was not known to any of the participants). It seems that reasoning outside of the technological or business context in which one is immersed, is sometimes actually quite difficult.

This opens up the question of whether for program comprehension experiments, one needs to select experts in the field, people that are aware of the particular difficulties faced, or novice users, who are not familiar with all of the problems in the area and that have a no bias when it comes to experience with potential difficulties in understanding systems.

6.3 Threats to validity

In this section we present a discussion of how the results of our experiment might be challenged. We discuss internal validity, namely whether the participants might have been directly affected by factors we did not consider, and external validity, meaning whether our results are generalizable.

6.3.1 Internal Validity

One possible threat to the internal validity could have been related to a lack in the participants' competence. However, in a short pre-study interview we did gauge for their experience with SOA and we established that all participants had at least 2 years of experience with developing SOA based systems. This, as we noted in Section 6.2, might have been both an advantage and a disadvantage. While experience with SOA is welcomed, we theorize it may have also been a source for bias towards a particular type of SOA system.

Participants might have been inclined to rate the tool more positively than they actually value it, because they might have felt this was the more desirable answer. We mitigated this concern by indicating to participants that only honest answers were valuable. The good mix of answers we have obtained is another indication of the participants answering the questions truthfully.

6.3.2 External Validity

Our main concern with external validity has to do with the performance impact of our approach. We are particularly concerned with the performance overhead added by the data collection step. However, the data collection is supported by a robust framework used by a company with a large web services infrastructure (eBay).

The storage requirement of our approach is also quite high [24]. In a system with a large traffic of web service requests, a large amount of storage space is required in order to maintain the system's state over time. This threat can be mitigated by using compression techniques, e.g., as applied in the Compact Trace Format (CTF) [12].

The applicability of our results to other, larger systems is also a possible threat. We tried to mitigate this by using Spicy Stonehenge, which, despite its small size, contains all the ingredients of an industrial system, including complex interactions between services and a well-specified domain.

7. RELATED WORK

In general, the increased maintenance complexity of SOA-based systems has been acknowledged and emphasized by Lewis and Smith [17], requiring for instance, impact analysis for an unknown set of users, or increased number of externally accessed services to be considered in maintenance. These are asking for a readjustment of current maintenance practice for SOA-based systems at large.

For this article, we started by analyzing the survey of Cornelissen et al. [5] on program comprehension through dynamic analysis, which, among others, lists the work of De Pauw et al. [6]. They describe a web services navigator for generating service topologies, and focus on detecting incorrect implementations of business rules and "excessively chatty" communications. Our approach is different w.r.t. two significant improvements to the service topology: it allows to identify the method of an invoked service plus its invoking client, and it includes the time dimension providing a historic view on the topology.

White et al. [23] present a dynamic analysis approach to aid the maintenance of SOA-based composite applications where they propose using a feature sequence viewer to recover sequence diagrams from such systems. This approach, however, does not seem to provide a basis for understanding the topology of a running service-oriented system and rather focuses on mapping features to software artifacts.

In other work, White et al. [22] investigated the information of developers during the maintenance of SOA-based systems. They established that the first question a maintainer must ask is "how does the software work *now*?"

Finally, we investigated citations to the aforementioned papers, in particular the systematic survey by Cornelissen et al. [5] in order to find more recent additions to the body of knowledge. Unfortunately, this search has yielded no extra relevant related work.

8. CONCLUSION

In this article, we investigate how the analysis of the users and versions in a multi-tenant system can help its understanding for maintenance purposes. More specifically, our contributions are:

- The runtime topology augmented with the time dimension.
- The runtime topology filtered by user and service version.
- Serviz, an open-source implementation of this approach.
- A field user study with 4 participants from two software engineering companies in order to evaluate the effectiveness of such an approach.

We now summarize how our approach and the tool address our original research questions formulated in Section 1:

RQ1 *Does the user information added to the runtime topology help in the understanding of SOA-based systems?* Our participants agree that having this information available helps in the understanding of a SOA-based system, with some remarks regarding automating the identification of important highlights (e.g. which users used the system the most in a specific period).

RQ2 *Does the service version information added to the runtime topology help in the understanding of SOA-based systems?* On what concerns service version information, on average all but one participant agreed that this having this information at hand makes understanding highly dynamic SOA-based systems easier. On systems where the runtime topology is less prone to change, the usefulness might prove to be diminished.

RQ3 *Do usage graphs help to understand SOA-based systems?* For our last research question, the participants generally agreed that time-based graphs add value to understanding SOA-systems and their maintenance. Furthermore, one of the companies is creating a prototype tool which offers a similar set of features, highlighting the relevance and usefulness of such a feature.

Future work

From our interview, the participants noted several aspects which we would like to improve as future work. An improvement we plan to add to our runtime topology deals with automatic identification of useful data. For example, the participants claimed that identifying peaks of high and low usage for a particular service costs a significant amount of time. Similarly, a common remark was the lack of a service usage referential. This means the participants had to manually calculate the usage of a particular service by dividing the number of requests by the specific time interval chosen. This is something which was also suggested as beneficial, should it be automated.

Another aspect we would like to further study is an automated dead code detection for service oriented systems. This

is particularly important in the context of multi-tenancy where specific versions are tailored for a particular user, who might in the future migrate to another version.

Our ultimate goal is to keep on developing Serviz, make it more user-friendly and improve its visualization. Some participants complained the graph did not retain its position every time the dates were changed, which caused great frustration.

9. ACKNOWLEDGMENTS

The authors would like to acknowledge NWO for sponsoring this research through the Jacquard ScaleItUp project. Also many thanks to the participants in our user study and their companies Adyen and Exact.

10. REFERENCES

- [1] C.-P. Bezemer and A. Zaidman. Multi-tenant saas applications: maintenance dream or nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 88–92. ACM, 2010.
- [2] G. Canfora and M. Di Penta. New frontiers of reverse engineering. In *Future of Software Engineering (FOSE)*, pages 326–341. IEEE CS, 2007.
- [3] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.
- [4] T. A. Corbi. Program understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.
- [5] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Trans. Software Eng.*, 35(5):684–702, 2009.
- [6] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. F. Morar. Web services navigator: Visualizing the execution of web services. *IBM Systems Journal*, 44(4):821–846, 2005.
- [7] K. A. Ericsson and H. A. Simon. How to study thinking in everyday life: Contrasting think-aloud protocols with descriptions and explanations of thinking. *Mind, Culture, and Activity*, 5(3):178–186, 1998.
- [8] T. Espinha, C. Chen, A. Zaidman, and H.-G. Gross. Maintenance research in SOA — towards a standard case study. In *Proc. of the Conf. on Software Maintenance and Reengineering (CSMR)*, pages 391–396. IEEE CS, 2012.
- [9] T. Espinha, A. Zaidman, and H.-G. Gross. Understanding the runtime topology of service-oriented systems. In *Proc. of the Working Conf. on Reverse Engineering (WCRE)*, pages 187–196. IEEE CS, 2012.
- [10] R. Fang, L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du. A version-aware approach for web service directory. In *IEEE International Conference on Web Services (ICWS)*, pages 406–413. IEEE CS, 2007.
- [11] N. Gold, C. Knight, A. Mohan, and M. Munro. Understanding service-oriented software. *IEEE Software*, 21(2):71–77, 2004.
- [12] A. Hamou-Lhadj and T. C. Lethbridge. A metamodel for the compact but lossless exchange of execution traces. *Software and System Modeling*, 11(1):77–98, 2012.
- [13] K. Holtzblatt and S. Jones. Human-computer interaction. chapter Conducting and analyzing a contextual interview (excerpt), pages 241–253. Morgan Kaufmann, 1995.
- [14] M. Kajko-Mattsson, G. A. Lewis, and D. B. Smith. Evolution and maintenance of soa-based systems at sas. In *Proc. Hawaii Int’l Conf. on Systems Science (HICSS)*, page 119. IEEE CS, 2008.
- [15] T. Kwok, T. Nguyen, and L. Lam. A software as a service with multi-tenancy support for an electronic contract management application. In *Proc. Int. Conf. on Services Computing (SCC)*, pages 179–186. IEEE, 2008.
- [16] M. M. Lehman and L. A. Belady. *Program Evolution: Processes of Software Change*. Academic Press, 1985.
- [17] G. Lewis and D. Smith. Service-oriented architecture and its implications for software maintenance and evolution. In *Proceedings Frontiers of Software Maintenance*, pages 1–10. IEEE CS, 2008.
- [18] N. Matthijssen, A. Zaidman, M.-A. Storey, I. Bull, and A. van Deursen. Connecting traces: Understanding client-server interactions in ajax applications. In *Proc. 18th Int. Conf. on Program Comprehension (ICPC)*, pages 216–225. IEEE CS, 2010.
- [19] J. Sillito, G. C. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *IEEE Trans. Softw. Eng.*, 34(4):434–451, 2008.
- [20] S. E. Sim and R. C. Holt. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In *Proc. of the International Conference on Software Engineering (ICSE)*, pages 361–370. IEEE CS, 1998.
- [21] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In *International Conference on e-Business Engineering (ICEBE)*, pages 94–101. IEEE, 2008.
- [22] L. White, N. Wilde, T. Reichherzer, E. El-Sheikh, G. Goehring, A. Baskin, B. Hartmann, and M. Manea. Understanding interoperable systems: Challenges for the maintenance of soa applications. pages 2199–2206. IEEE CS, 2012.
- [23] L. J. White, T. Reichherzer, J. Coffey, N. Wilde, and S. Simmons. Maintenance of service oriented architecture composite applications: static and dynamic support. *J. Softw. Maint. Evol.: Res. Pract.*, 25(1):97–109, 2013.
- [24] A. Zaidman, B. Du Bois, and S. Demeyer. How webmining and coupling metrics improve early program comprehension. In *Proc. Int’l Conf. on Program Comprehension (ICPC)*, pages 74–78. IEEE CS, 2006.
- [25] A. Zaidman, N. Matthijssen, M.-A. D. Storey, and A. van Deursen. Understanding ajax applications by connecting client and server-side execution traces. *Empirical Software Engineering*, 18(2):181–218, 2013.