

# Pre-Proceedings of the ES OCC 2018 Workshops

## Joint CloudWays and OptiMoCS Workshop

Vasilios Andrikopoulos, Nane Kratzke, Zoltán Ádám Mann, Claus Pahl  
(Editors)

## 14th International Workshop on Engineering Service- Oriented Applications and Cloud Services

Luciano Baresi, Willem-Jan van den Heuvel, Andreas S. Andreou, Guadalupe Ortiz, Christian Zirpins, George Feuerlicht, Winfried Lamersdorf (Editors)

### **Abstract**

This volume contains the papers presented at the WESOACS and Joint CloudWays and OptiMoCS workshops associated with the 7th European Conference on Service-Oriented and Cloud Computing, ES OCC 2018. The workshops were held in Como, Italy, on 12th September 2018. The workshops covered specific topics in service-oriented and cloud computing-related domains:

- 4th International Workshop on Cloud Migration and Architecture (CloudWays 2018)
- 1st International Workshop on Optimization in Modern Computing Systems (OptiMoCS 2018)
- 14th International Workshop on Engineering Service-Oriented Applications and Cloud Services (WESOACS 2018)

All papers presented at the workshops were selected through a rigorous review process, in which each submission was reviewed by at least three members of the workshops' program committees.

We as the workshop chairs would like to thank all authors for their submissions, and the reviewers for their work.

## **WESOACS**

Towards a Generalizable Comparison of the Maintainability of Object-Oriented and Service-Oriented Applications 55

*Justus Bogner, Bhupendra Choudhary, Stefan Wagner and Alfred Zimmermann*

Implementation of a Cloud Services Management Framework 67

*George Feuerlicht and Thai Hong Tran*

Decentralized Billing and Subcontracting of Application Services for Cloud Environment Providers 79

*Wolf Posdorfer, Julian Kalinowski, Heiko Bornholdt and Winfried Lamersdorf*

May Contain Nuts: The Case for API Labels 90

*Cesare Pautasso and Erik Wilde*

On Limitations of Abstraction-Based Deadlock-Analysis of Service-Oriented Systems 102

*Mandy Weissbach and Wolf Zimmermann*

# Towards a Generalizable Comparison of the Maintainability of Object-Oriented and Service-Oriented Applications

Justus Bogner<sup>1,2</sup>, Bhupendra Choudhary<sup>2</sup>, Stefan Wagner<sup>2</sup>, and Alfred Zimmermann<sup>1</sup>

<sup>1</sup> University of Applied Sciences Reutlingen, Germany  
{justus.bogner,alfred.zimmermann}@reutlingen-university.de  
<sup>2</sup> University of Stuttgart, Germany  
{justus.bogner,stefan.wagner}@informatik.uni-stuttgart.de  
bhupendra.choudhary@gmx.de

**Abstract.** While there are several theoretical comparisons of Object Orientation (OO) and Service Orientation (SO), little empirical research on the maintainability of the two paradigms exists. To provide support for a generalizable comparison, we conducted a study with four related parts. Two functionally equivalent systems (one OO and one SO version) were analyzed with coupling and cohesion metrics as well as via a controlled experiment, where participants had to extend the systems. We also conducted a survey with 32 software professionals and interviewed 8 industry experts on the topic. Results indicate that the SO version of our system possesses a higher degree of cohesion, a lower degree of coupling, and could be extended faster. Survey and interview results suggest that industry sees systems built with SO as more loosely coupled, modifiable, and reusable. OO systems, however, were described as less complex and easier to test.

**Keywords:** Maintainability · Service Orientation · Object Orientation · Metrics · Experiment · Survey · Interviews

## 1 Introduction

The ability to quickly and cost-efficiently change applications and services due to new or redacted requirements is important for any company relying on custom software. The associated quality attribute is maintainability: the degree of effectiveness and efficiency with which software can be changed [5], e.g. to adapt or extend it. The introduction of Object Orientation (OO) lead to maintainability-related benefits like encapsulation, abstraction, inheritance, or increased support for modularization [3]. In today's enterprise world, however, systems built on Service Orientation (SO) are increasingly more common. By introducing a higher level of abstraction, Service-Based Systems (SBSs) consist of loosely coupled distributed components with well defined technology-agnostic interfaces [7]. SO

aims to promote interoperability, reuse of cohesive functionality at a business-relevant abstraction level, and encapsulation of implementation details behind published interfaces [4].

So while Service Orientation seems to surpass Object Orientation w.r.t. maintainability from a theoretical point of view, this comparison is very hard to generalize in a practical setting. Developers can build systems of arbitrary quality in both paradigms, although the inherent properties of both paradigms may make it easier or harder to build well maintainable systems. Very little empirical research exists on the topic of comparing the maintainability of OO and SO (see Sect. 2). Results from such studies can bring valuable insights into the evolution qualities of these two paradigms. Research in this area can also highlight potential deficiencies and weaknesses, which helps raising awareness for developers as well as providing decision support for choosing a paradigm for a project.

This is why we conducted a study to compare the maintainability of object-oriented and service-oriented applications from different perspectives. For a practical empirical point of view, we constructed two functionally equivalent systems (one based on OO and the other on SO) and compared them with metrics as well as by means of a controlled software development experiment. To gain insight into software professionals' subjective estimation of the two paradigms, we conducted an industry survey as well as expert interviews. In the remainder of this paper, we first introduce related work in this area. Then we present the details of our 4-part study including the methods, results, and limitations. Lastly, we conclude by summarizing our results and putting them into perspective.

## 2 Related Work

A small number of scientific publications exists that compare Service Orientation and Object Orientation. In 2005 when SBSs were still very young, Baker and Dobson [1] published a theoretical comparison of Service-Oriented Architecture (SOA) and Distributed Object Architectures (DOA) based on literature and personal experience. Their comparison is very high-level and not focused specifically on maintainability. While they highlight a large number of similarities, they also point out the more coarse-grained interfaces of SOA that lead to simplified communication and less cognitive overhead for developers of service consumers. Moreover, they point out the missing notion of inheritance and interface specialization in SOA, which they acknowledge as initially less complex, but potentially limiting in the long term.

Stubbings [10] provided another theoretical comparison that also emphasizes the direct line of evolution from OO to SO. Beneficial OO concepts like encapsulation and reuse have been adapted to a higher abstraction level in Service Orientation that is closer to the business domain. He further assessed the structural and technological complexity to be higher in a system based on Service Orientation. Concerning communication, he reported the focus for OO to be primarily internal while SO would be more aimed at external interoperability.

One of the few empirical studies on the subject was performed by Perepletchikov et al. [8] on two versions of a fictional Academic Management System (one service-oriented version, the other one object-oriented). To compare the maintainability of the two, they employed traditional source code metrics like *Lines of Code*, *Cyclomatic Complexity*, as well as the OO metrics suite from Chidamber and Kemerer. They focused on the structural properties size, complexity, coupling, and cohesion. As findings, they reported that the SO version provides better separation of business and implementation logic and a lower degree of coupling. The OO system, however, would be overall less complex.

Lastly, Mansour and Mustafa [6] conducted a similar empirical study. They constructed a service-oriented version of an existing OO Automated Teller Machine system and compared the two versions with a set of metrics, very similar to the ones in [8]. They reported that the SO version of their system inhibited a higher degree of reusability and a lower degree of coupling while the complexity of the OO version was lower. Additionally, they described difficulties when trying to apply OO metrics to a Service-Based System and advocated the need for a set of service-oriented maintainability metrics.

Existing studies are either of a theoretical nature or solely focused on metrics. While the presented empirical studies provide first valuable support for a comparison with metrics, they also reported difficulties due to a lack of mutually applicable metrics. Not all OO metrics can be used for SBSs. Moreover, additional metric evaluations with other systems will be of value while new approaches can bring different perspectives to the discussion.

### 3 Study Design

Based on the results and lessons learned of the related work, we therefore conducted a study with four different parts. First, we constructed a service-oriented and an object-oriented version of a simple Online Book Store (OBS) that provided functionality to register as a user as well as to browse and order books. The service-oriented version was implemented with RESTful NodeJS services using the Express framework<sup>3</sup> and an Angular frontend<sup>4</sup> while the object-oriented version is a Java monolith relying on JavaServer Pages (JSP) as a web UI. These two systems were compared using a set of **coupling and cohesion metrics**. To respect the two system versions, we needed metrics that can be applied both to service- as well as object-oriented systems. This is often difficult to achieve, since coupling and cohesion metrics are usually designed for either of the two paradigms. We therefore chose two metrics for each structural property that could be adapted to be mutually applicable.

For coupling, we chose *Absolute Importance of the Service* (AIS) and *Absolute Dependence of the Service* (ADS). Both have been specifically designed for SBSs and represent the number of clients invoking a service (AIS) and the number of

<sup>3</sup> <https://expressjs.com>

<sup>4</sup> <https://angular.io>

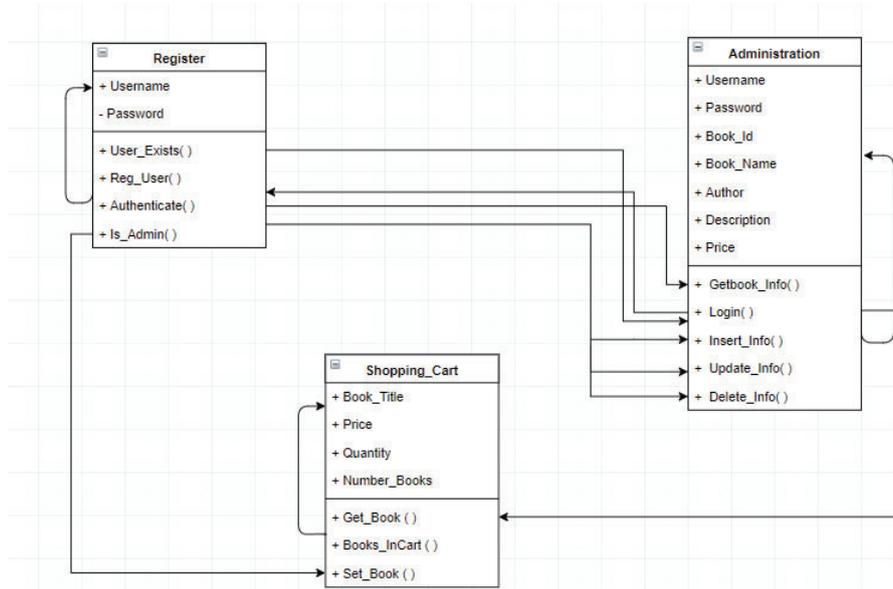


Fig. 1. Object-Oriented Version of OBS

other services a service depends on respectively (ADS) [9]. They can be easily adapted to object-oriented systems by substituting *services* with *classes*.

For cohesion, we selected two object-oriented metrics, namely *Tight Class Cohesion* (TCC) and *Loose Class Cohesion* (LCC) [2]. These metrics attempt to measure the relatedness of class functionality based on common class attributes that the methods operate on. TCC represents the relative number of directly connected methods while LCC also includes indirectly connected methods (via other intermediate methods). To adapt these metrics to a service-oriented context, class methods are substituted by service operations.

While the majority of maintainability metrics use structural properties as a proxy, industry is really interested in something else: how fast can changes or features be implemented for the system? To account for this, the same systems were used in a **controlled experiment**. Software practitioners had to implement search functionality for books while the time was measured. We then analyzed whether the version made a noticeable difference. 8 software developers participated in the experiment, four per system version of OBS. 7 of the 8 developers were from Germany. They had an average of  $\sim 4.1$  years of experience (OO AVG: 4.5 years, SO AVG: 3.75 years). All of them had worked with their respective paradigm before. We measured the time necessary to complete the exercise as well as the changed Lines of Code for the backend part.

To complement these two empirical approaches, we also conducted an **industry survey** to capture the general sentiment of developers towards the two

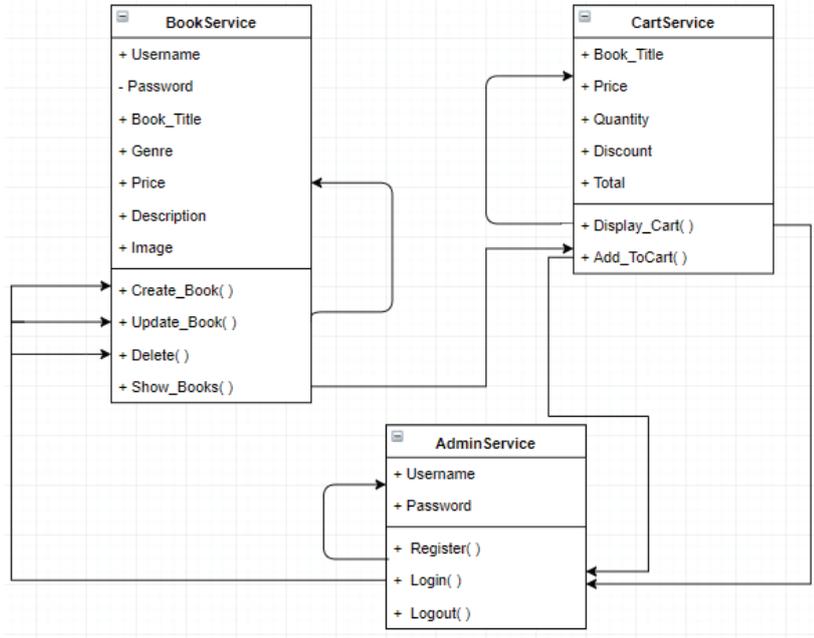


Fig. 2. Service-Oriented Version of OBS

paradigms. Software professionals filled out an online questionnaire where they were asked to compare structural and maintainability-related properties of the two paradigms based on their personal experience. 32 participants completed our web-based questionnaire that was distributed via personal industry contacts, mailing lists, and social media. The survey was hosted from 2018-04-19 until 2018-05-06 and consisted of 12 questions, mostly with Likert scale answers. Most participants were from Germany and India and all had at least three years of professional experience. They had to comment on the average condition of different structural properties (e.g. coupling) and subquality attributes of maintainability in SW projects based on either OO or SO. Lastly, they had to answer some questions where they ranked the three paradigms *Object Orientation*, *Service Orientation*, and *Component-Based* for similar attributes.

As a more in-depth follow-up to the survey, we conducted **qualitative interviews** with several experts to complement the broader scope of the survey and to dive more deeply into some of the topics. Similar to the survey, we also asked for their personal experience and preference w.r.t. the maintainability of the two paradigms under study. This was the fourth and final part of our study. All 8 experts had an IT or Engineering background and had previously worked with object-oriented as well as service-oriented systems. 7 of the 8 experts were older than 30 years, i.e. had considerable professional experience. The interviews started with an introduction of the two OBS versions and a discussion

about their strengths and weaknesses. This was followed by similar questions as in the survey about properties of the two paradigms and the participants' experience.

Please refer to our GitHub repository for the source code of the systems as well as the detailed survey questions and results <sup>5</sup>.

## 4 Results

For the metric-based part of the study, we measured all four **component-level metrics** for both the object-oriented (Fig. 1) and the service-oriented version (Fig. 2) of the Online Book Store (OBS). Since each version of the system includes three components (services or classes respectively), we have a total of 12 measurements (see Table 1). When looking at the AVG values per version and metric (see Fig. 3), we can see that the service-oriented version overall has slightly better values, i.e. on average lower coupling and higher cohesion per component.

**Table 1.** Coupling and Cohesion Metric Values per Component

	Component	AIS	ADS	TCC	LCC
OO Version	<b>Administration</b>	1	2	0.00	0.40
	<b>Register</b>	1	2	0.16	0.50
	<b>Shopping_Cart</b>	2	0	0.33	0.33
SO Version	<b>AdminService</b>	1	1	0.67	0.67
	<b>BookService</b>	1	1	0.33	0.50
	<b>CartService</b>	1	1	1.00	1.00

During the **controlled experiment**, it took less time and effort to extend the service-oriented version of OBS (see Fig. 4). The mean duration for the SO version was 0.8 h while it was 0.99 h for the OO version. Respectively, the mean effort was 7.25 LoC for SO and 12.5 LoC for OO. When analyzing the significance of the mean differences in our sample with an unpaired t-test, we found two-tailed p-values smaller than 0.05 (p-value<sub>duration</sub>: 0.0479, p-value<sub>effort</sub>: 0.005).

The following part highlights the results of the **survey questions**. For Likert scale question, we also present the aggregated score per paradigm (Strongly Disagree: -2, Disagree: -1, Neutral: 0, Agree: 1, Strongly Agree: 2).

Question: *In my experience, software based on <paradigm> has a comparatively low degree of **coupling**.*

<sup>5</sup> <https://github.com/xJREB/research-oo-vs-so>

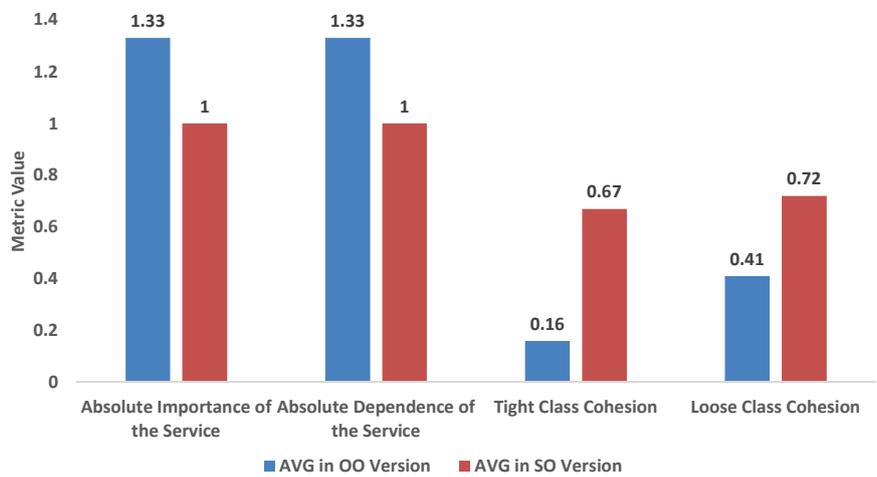


Fig. 3. Average Coupling and Cohesion Metric Values per Version



Fig. 4. Experiment: Duration per Participant

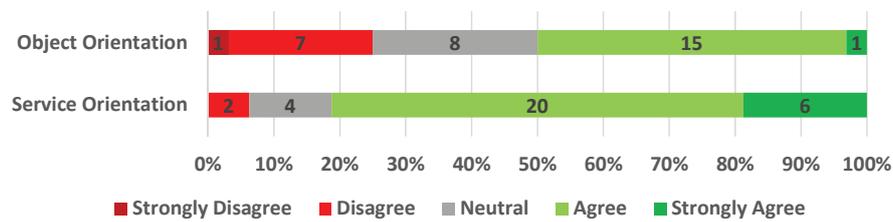


Fig. 5. Question: In my experience, software based on <paradigm> has a comparatively low degree of coupling.

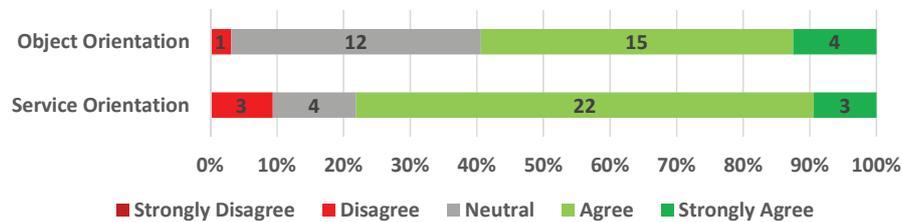
For coupling, participants clearly favored Service Orientation (score: 30) over Object Orientation (score: 8). Over 80% reported that service-oriented systems were in their experience of a more loosely coupled nature while only 50% reported the same for object-oriented systems (see Fig. 5). This result was to be expected, since loose coupling and the reduction of dependencies is a major driver in SBSs.

Question: *In my experience, software based on <paradigm> facilitates a comparatively high degree of **cohesion**.*

When it came to cohesion, the results were less decisive (SO: 18, OO: 14). Overall, roughly 13% more participants agreed with this statement for Service Orientation (SO: ~63%, OO: 50%). This does not seem to be a lot, when we consider the prevalence of the “cohesive services grouped around business capabilities” theme in an SOA and especially in a Microservices context.

Question: *In my experience, software based on <paradigm> promises a significant extent of **reusability**.*

Participants reported higher reusability for their service-oriented software than for their object-oriented software. While the scores were pretty even (SO: 25, OO: 22), ~78% of participants agreed to this statement for SO while only ~59% agreed for OO. Absolute scores are so close because two more people disagreed for SO and one more strongly agreed for OO (see Fig. 6). Overall, these results seem to support the SO principle of business-relevant reuse granularity.

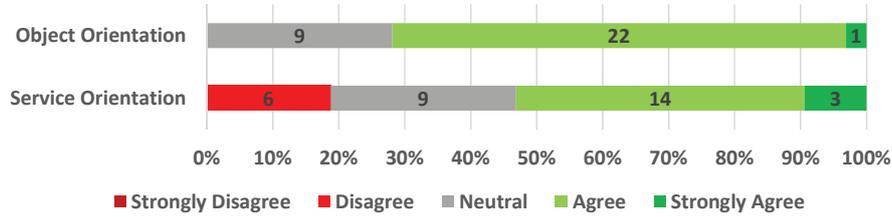


**Fig. 6.** Question: In my experience, software based on <paradigm> promises a significant extent of **reusability**.

Question: *In my experience, software based on <paradigm> reduces the complexity of **testing**.*

In the case of testability, Object Orientation (score: 24) was seen as more beneficial than Service Orientation (score: 14) to reduce complexity. Roughly 72% of participants agreed with this statement for OO while only ~53% agreed for SO together with 6 disagreements (see Fig. 7). This is the first category where OO decisively wins out in the opinion of participating developers.

Lastly, developers were asked to rank the three paradigms *Object Orientation*, *Service Orientation*, and *Component-Based* from their experience for three further properties: modifiability, encapsulation/abstraction, and size/complexity. Ranking a paradigm first provided three points, ranking it second provided two, ranking it last provided one point respectively. The results (see Table 2) indi-



**Fig. 7.** Question: In my experience, software based on <paradigm> reduces the complexity of **testing**.

**Table 2.** Question: In your experience, which of the three paradigms provides on average the most favorable degree of <attribute>?

	Object Orientation	Component-Based	Service Orientation
Modifiability	63	43	86
Encapsulation and Abstraction	58	43	85
Size and Complexity	74	39	73

cate that participants experienced systems based on Service Orientation as more modifiable and with a better degree of encapsulation and abstraction as for the other two paradigms. For size and complexity, however, participants reported that they believed the manageability of these properties to be roughly equal for OO and SO, with OO winning out by one point.

We compiled results from the **qualitative interviews** in several areas. For the topic of *modifiability*, 5 of the 8 experts reported that on average in their experience service-oriented systems are more beneficial than object-oriented ones when it comes to evolving already developed systems. Participants emphasized the advantages of service-based modularity, which would increase independence in the system and reduce costs in the long run. Some experts highlighted that SO is more convenient when requirements frequently change.

Concerning *complexity*, most experts indicated based on their past software projects that systems based on Object Orientation are on average less complex than SBSs from a structural and technological point of view. They also mentioned mature tool support in the field of object-oriented SW development that would ease some of the difficulties. In the service-oriented space, however, tool support would be lacking.

When comparing the average *analyzability* of the two paradigms, the majority of participants favored Service Orientation over Object Orientation. The structure of the system would be easier to grasp when referring to services as coarse-grained components. Moreover, experts experienced less dependencies in SBSs, which also helped to comprehend the structure of a system.

Lastly, in addition to the lack of mature tool support for Service Orientation, participants reported the danger of ripple effects when changing services, especially with service interface changes that require updates of all service consumers. Some experts also stressed that Object Orientation was a valuable paradigm to be used for the inner low-level design of single services and that it would nicely complement the service-based high-level architecture of a system. So the choice would not always be either Service or Object Orientation.

## 5 Threats to Validity

Several things have to be mentioned to put our results into appropriate perspective. For the **metric-based evaluation**, the tested systems were artificially constructed and are not real industry or open source systems. While we tried to design and implement them as close to a real use case as possible, we also needed something of manageable size and complexity, which may impact the generalizability of the comparison (e.g. the AVG metric values were computed from only three components). The chosen technology for both versions may also be a limitation. Results with other programming languages or frameworks could be different. Moreover, we only used a small number of metrics and targeted only two structural properties (coupling and cohesion). Other metrics, e.g. for size or complexity, could have yielded additional insights, but were neglected due to project time constraints. Finally, we calculated the metric values manually due to missing tool support. Since the systems are of limited size and we double-checked each value, the error probability should still be very small.

In the case of the **controlled experiment**, the same limitations of the constructed systems as described above hold true. The two different programming languages (Java and NodeJS/JavaScript) also limit the comparability of the LoC effort. Additionally, we only had a small number of participants. Potentially different development experience and skill levels could not be accounted for when assigning the participants to the two versions of OBS. Lastly, the experiment consisted of only one exercise, which can only test the modifiability of certain parts of the system.

As with most **quantitative surveys**, a number of limitations have to be mentioned. First, the number of participants (32) only provides limited generalizability, as a different population subset may have different views on the subject. Moreover, we could not guarantee that the participating developers indeed had sufficient experience with all three software paradigms. Lastly, the subjective estimation of the inherent qualities of a paradigm may be skewed by a particularly bad experience with a suboptimally designed system. Overall, it is important to keep in mind that personal preference of developers is not necessarily of a rational nature.

As opposed to our survey participants, we could select our **interview experts** based on their experience with the two paradigms under evaluation, at least up to a certain degree. However, there is still a chance that some experts were less proficient with one of the paradigms or were heavily influenced by one

specific project of theirs. Moreover, there is a chance that we slightly influenced the experts by posing questions that should direct the conversation to the properties under evaluation. Lastly, our interviews were conducted and analyzed in a fairly loosely structured manner without a rigorous methodology.

## 6 Conclusion

To provide additional evidence for a generalizable comparison of the maintainability of Service Orientation and Object Orientation, we conducted a study with four parts: a metric-based comparison of two functionally equivalent systems (one SO and one OO version); a controlled experiment where practitioners had to extend the same systems; an industry survey with comparative questions about OO and SO; and expert interviews as a more in-depth follow-up to the survey.

The empirical results indicate that the service-oriented version of our Online Book Store system consists of more cohesive and more loosely coupled components and could also be extended faster and with less effort (LoC) by experiment participants. Survey and interview results seem to go in the same direction: industry professionals experienced higher modifiability, lower degrees of coupling, higher reusability, and stronger encapsulation and abstraction in their service-oriented projects. For their average object-oriented systems, however, they reported comparatively lower complexity and better testability.

While these results can aid in the decision process for a paradigm and can highlight important maintainability-related focus points when designing systems with either paradigm, it is still important to remember that we can build software of arbitrary quality in both paradigms. Moreover, Object Orientation can be a useful complement for the inner architecture of services.

**Acknowledgments** This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program “Services Computing” (<http://www.services-computing.de/?lang=en>).

## References

1. Baker, S., Dobson, S.: Comparing Service-Oriented and Distributed Object Architectures. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3760 LNCS, pp. 631–645 (2005)
2. Bieman, J.M., Kang, B.K.: Cohesion and reuse in an object-oriented system. In: Proceedings of the 1995 Symposium on Software reusability - SSR '95. pp. 259–262. ACM Press, New York, New York, USA (1995)
3. Booch, G.: Object Oriented Analysis & Design with Application. Pearson Education (2006)
4. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)

5. International Organization For Standardization: ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Tech. rep. (2011)
6. Mansour, Y.I., Mustafa, S.H.: Assessing Internal Software Quality Attributes of the Object-Oriented and Service-Oriented Software Development Paradigms: A Comparative Study. *Journal of Software Engineering and Applications* 04(04), 244–252 (2011)
7. Papazoglou, M.: Service-oriented computing: concepts, characteristics and directions. In: *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials* (Cat. No.03CH37417). pp. 3–12. IEEE Comput. Soc (2003)
8. Perepletchikov, M., Ryan, C., Frampton, K.: Comparing the impact of service-oriented and object-oriented paradigms on the structural properties of software. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3762 LNCS, 431–441 (2005)
9. Rud, D., Schmietendorf, A., Dumke, R.R.: Product Metrics for Service-Oriented Infrastructures. In: *IWSM/MetriKon* (2006)
10. Stubbings, G.: Service-Oriented and Object-Oriented: Complementary Design Paradigms. *SPARK: The ACES Journal of Postgraduate Research* 1 (2010)