SICS Software-Intensive Cyber-Physical Systems
https://doi.org/10.1007/s00450-019-00402-z

SPECIAL ISSUE PAPER

Using architectural modifiability tactics to examine evolution qualities of Service- and Microservice-Based Systems

An approach based on principles and patterns

Justus Bogner[1,2] · Stefan Wagner[2] · Alfred Zimmermann[1]

Abstract
Software evolvability is an important quality attribute, yet one difficult to grasp. A certain base level of it is allegedly provided by Service- and Microservice-Based Systems, but many software professionals lack systematic understanding of the reasons and preconditions for this. We address this issue via the proxy of architectural modifiability tactics. By qualitatively mapping principles and patterns of Service-Oriented Architecture (SOA) and Microservices onto tactics and analyzing the results, we cannot only generate insights into service-oriented evolution qualities, but can also provide a modifiability comparison of the two popular service-based architectural styles. The results suggest that both SOA and Microservices possess several inherent qualities beneficial for software evolution. While both focus strongly on loose coupling and encapsulation, there are also differences in the way they strive for modifiability (e.g. governance vs. evolutionary design). To leverage the insights of this research, however, it is necessary to find practical ways to incorporate the results as guidance into the software development process.

Keywords Modifiability · Architectural tactics · Patterns · Service-Based Systems · SOA · Microservices

1 Introduction

Several modern enterprise software systems are expected to quickly incorporate modifications or extensions should changes in functional or non-functional requirements arise. Quality attributes related to software evolution like maintainability or evolvability are often hard to control, especially in long-living systems that are continuously changed and extended.

Service-Based Systems [22] and especially the more recent variant Microservice-Based Systems [21] allegedly provide a high base degree of evolvability, since their architecture is supposed to focus on beneficial principles like loose coupling, high cohesion, encapsulation, reuse, and composition. There is evidence that software professionals self-reportedly have a strong belief in the inherent qualities of these types of systems [3, 30], to such a degree in fact that they may even reduce maintainability control actions [8]. It is very hard, however, to generalize the evolution qualities of systems based on service orientation and compare them to e.g. object-oriented systems. Generalizable empirical support for this is difficult to provide (see for example [5, 25, 31]). A blind belief in the inherent modifiability of a Service-Based System—without having a clear understanding of the factors that influence this quality attribute—can lead to violations of important principles and therefore negatively impact software evolution. To prevent this, systematic knowledge of service-oriented evolution qualities and how they can be achieved and preserved is necessary.

Our approach to address this is based on relating general architectural modifiability tactics to (a) principles of Service- and Microservice-Based Systems and to (b) design

✉ Justus Bogner
justus.bogner@reutlingen-university.de

Stefan Wagner
stefan.wagner@informatik.uni-stuttgart.de

Alfred Zimmermann
alfred.zimmermann@reutlingen-university.de

[1] University of Applied Sciences Reutlingen, Reutlingen, Germany

[2] University of Stuttgart, Stuttgart, Germany

Published online: 08 February 2019

patterns for these systems. The result is a matrix that maps general modifiability guidelines into a service-oriented context and relates them to concrete patterns that provide realization blueprints. Practitioners can use these first results to deepen their understanding of the evolution qualities of their Service-Based Systems and implement suggested patterns to enhance modifiability. Moreover, our mapping provides a comparison between the two service-based architectural styles Service-Oriented Architecture (SOA) and Microservices and their different strategies to achieve evolvability.

2 Background

The following sections provide the technical background for this work, namely architectural tactics and service-oriented design patterns.

2.1 Architectural tactics

Quality attributes (sometimes also referred to as non-functional requirements) play a key role in the process of designing the architecture for a particular software system. To provide general techniques that support software architects in achieving quality attribute goals, the Carnegie Mellon Software Engineering Institute in Pittsburgh, PA, conceptualized so called *Architectural Tactics* [7]. These techniques exist for various quality attributes, e.g. availability, performance, or security. An architectural modifiability tactic therefore is a design decision or an architectural transformation that positively affects system properties related to modifiability with the final goal of reducing the time and effort necessary to introduce future changes [4]. As opposed to patterns, tactics are described in a more general and less complex way without much implementation guidance. Moreover, they are only related to a single quality attribute and are therefore not concerned with design trade-offs.

2.2 Service-oriented patterns

Design patterns are a popular way to capture working and battle-tested solutions to recurring design problems. They are documented within a certain context and in a technology-agnostic way that can serve as a blueprint for several slightly differing concrete implementations. Originating from Alexander's building pattern language [1], they quickly became popular in various computer science domains, especially software engineering and software architecture. While the object-oriented design patterns of the "Gang of Four" [16] may arguably be the most popular ones, there is also a large number of service-oriented patterns that have mostly been conceptualized for the context of Service-Oriented Architecture (SOA) [12, 13, 29]. Although there is the indication that a significant number of these patterns is also fully or partially applicable to Microservice-Based Systems [9], pattern catalogs specifically for the context of Microservices are currently created [28]. The relationship between patterns and quality attributes is often complex and empirical support for their effectiveness is controversial [2, 17]. Nonetheless, the effect of patterns on quality attributes is a topic of great interest [15, 19, 26], as clarity in this regard could greatly help software professionals to choose relevant patterns to achieve quality attribute goals.

3 Related work

Several publications on the usage of architectural tactics exist within the context of Service-Based Systems. Mcheick et al. [18] investigate design decisions related to quality attributes for such systems. They propose a lightweight manual design method called the Service-Oriented Architecture Design Method (SOADM) that takes functional requirements and quality attributes as input and produces an architectural model of the necessary services and their interactions. To ensure that quality attribute goals are achieved, architectural tactics are used to enrich business services with system-related components that should realize tactics (they use the availability tactic *Ping/Echo* as an example). The method and its output is very general so that the resulting architectural model seems to need substantial refinement before implementation can be started.

Mirandola et al. [20] propose an automatic adaptation process for Service-Based Systems that should balance functional and non-functional requirements. It is based on adaptation space exploration via metaheuristic search techniques and uses the Service Component Architecture (SCA) for modeling. As an alternative to the metaheuristic search, a method called "Application of Existing Design Solutions" is provided. This method uses architectural patterns and tactics to produce new design candidates. In the example, availability tactics (e.g. *Ping/Echo* and *Heartbeat*) are used to satisfy a reliability requirement by adding a Monitoring Component to the architecture.

Parvizi-Mosaed et al. [24] improve the data collection analysis phase of the service-oriented SASSY (Self-Architecting Software SYstems) framework with an automated method to analyze the availability of an architecture. They do this by means of identifying, composing, and evaluating the availability of architectural patterns and tactics in the system via probability distribution functions. To be machine-readable, the tactics and patterns are modeled in RBML (Role-Based Modeling Language). The method requires very specific inputs, assumes that the complete system is designed with patterns and tactics, and is solely applied to the quality attribute availability.

Similarly, the same authors [23] extend the Service-Oriented Modeling and Application (SOMA) methodology with a self-adaptive process to maintain architecture quality based on the MAPE-K model (Monitor service, Analyze service, Plan service, Execution service, Knowledge management service). In the context of the Monitor Service, architectural tactics are combined with patterns as *Customized Patterns* to address a quality attribute scenario. Tactics are again modeled in RBML and associated with feature models to make it possible to analyze how several tactics influence each other. To facilitate the quality evaluation via tactics as a proxy, it is first necessary to instantiate a knowledge repository, which seems very time-consuming.

Lastly, Capelli and Scandurra [10] present a framework (SCA-PatternBox) for designing and prototyping service-oriented applications, which is also based on the SCA standard. Architectural patterns and tactics are used in the process steps where the architecture is refined w.r.t. to quality requirements and where service interfaces are finalized. They provide Eclipse-based tool support for modeling and evaluating the application. Patterns and tactics are formally modeled in SCA-ASM (Service Component Architecture-Abstract State Machine) to enable verification.

4 Scope and research method

All presented approaches in Sect. 3 incorporate architectural tactics as a means to increase the quality of the system under consideration. However, the application of tactics stays either on a very high-level or—if patterns are used—is either not really directed towards a concrete implementation or fairly removed from a service-oriented context. Very few popular service-oriented patterns are mentioned and no data is shared publicly. Furthermore, most approaches require intensive preparation and modeling of tactics and/or patterns, which has to be done by someone with intricate knowledge of these entities and their relationships. Lastly and most importantly, modifiability—as one of the more elusive quality attributes which is difficult to generalize and quantify—is not explicitly discussed. Most approaches focus on availability, either with the scope of the complete method or for exemplifying the approach. This is why our goal with this paper is to provide systematic understanding of how modifiability tactics relate to a service-based context and what service-oriented patterns can be used to realize them when designing or evolving Service- and Microservice-Based Systems. This goal is framed by the following research questions:

RQ1: To what extent are architectural modifiability tactics related to principles of Service- and Microservice-Based Systems?

RQ2: What concrete design patterns from the domain of Service- and Microservice-Based Systems can be used to realize modifiability tactics?

RQ3: What general statements about the evolution qualities of the two service-based architectural styles can be derived from the results of RQ1 and RQ2?

To answer these questions, we first compiled a list of existing architectural modifiability tactics from the most prominent sources [4, 6, 7]. These tactics are presented in Fig. 1 and organized in three categories:

- **Increase Cohesion** (to localize changes)
- **Reduce Coupling** (to prevent ripple effects)
- **Defer Binding Time** (to increase flexibility)

For the other part of the mapping, we had to choose recognized principles of Service- and Microservice-Based Systems. For the first, we used the service-oriented design principles from Erl [11] while for the latter we relied on the Microservices characteristics described by Lewis and Fowler

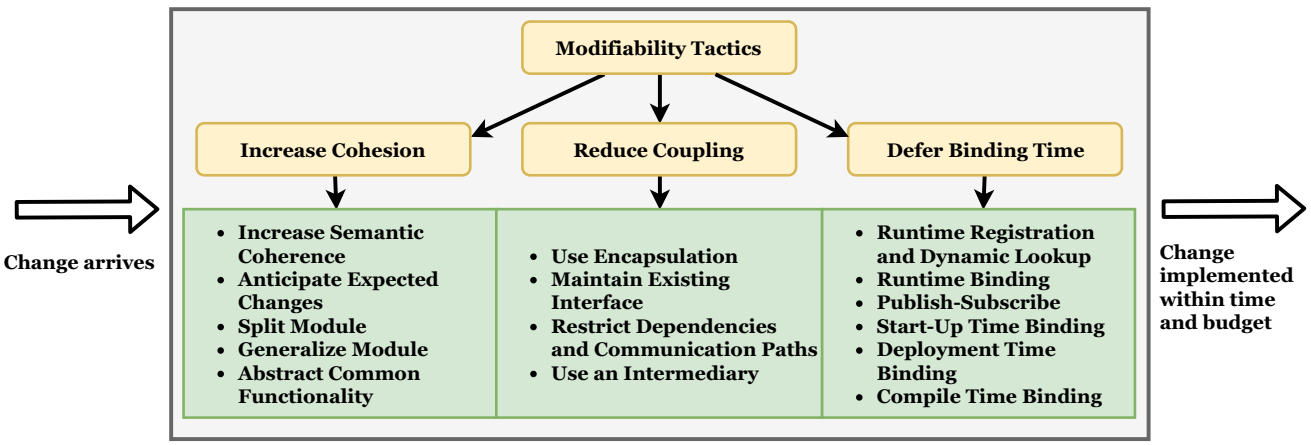

Fig. 1 Compiled list of architectural modifiability tactics based on [4, 6, 7]

Table 1 Service-oriented principles from [11]

Standardized service contract
Service loose coupling
Service abstraction
Service reusability
Service autonomy
Service statelessness
Service discoverability
Service composability

Table 2 Microservices principles from [14]

Componentization via services
Organized around business capabilities
Products, not projects
Smart endpoints, dumb pipes
Decentralization
Infrastructure automation
Design for failure
Evolutionary design

[14] (see Tables 1 and 2). Minor issues with this comparison are that a lot of Erl's service-oriented principles are certainly valid for Microservice-Based Systems as well. Moreover, the characteristics for the Microservices architectural style touch several new dimensions (e.g. operations, development, organization) that most "traditional" architectural styles (including SOA) do not consider. Nonetheless, we decided to explicitly include Microservices principles to also incorporate the other side of the service-oriented spectrum. This also has the added benefit of providing another perspective on the controversial "SOA vs. Microservices" debate (see e.g. [9, 27, 32]).

For the service-oriented patterns, we combined the SOA design pattern catalogs from Erl [12, 13] with the one from Rotem-Gal-Oz [29], since they have very little overlap and provide the most comprehensive essence of industry knowledge in this regard (118 patterns in total). Following the same reasoning as with the principles, we decided to include Microservices patterns as well. Since this is a very young and emerging topic, there are not a lot of options, results may be preliminary, and there certainly is controversy around the novelty of these patterns. We used Richardson's catalog (42 patterns) that is nearing completion at the time of writing [28]. While we are aware that all chosen sources for principles and patterns are not scientific peer-reviewed publications, we believe them to be valuable nonetheless. Moreover, they are quite popular in industry and most of them have been used and referred to in scientific publications.

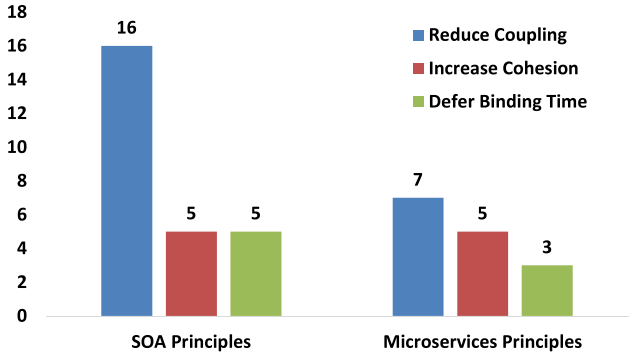

Fig. 2 Number of relations between tactic categories and principles

After completing the collection process, we first created a qualitative mapping between principles (for both Service- and Microservice-Based Systems) and tactics, i.e. which principles are related to or have an influence on which tactics? As the second step, we related patterns (again for both Service- and Microservice-Based Systems) to the tactics, i.e. which patterns can be used to realize which tactics? The mapping results were then documented to capture the underlying evolvability-related knowledge and analyzed for interesting distributions or anomalies. Please refer to our GitHub repository for the complete data set.[1]

5 Results (RQ1 and RQ2)

When mapping the 8 service-oriented principles from Erl and the 8 Microservices principles from Fowler onto the modifiability tactics and looking at the results quantitatively (see Fig. 2), some interesting points immediately stand out. First, while a maximum of 120 relations would be possible per service-oriented paradigm (8 principles * 15 tactics), we identified only 26 for SOA and only 15 for Microservices. In both paradigms, "Reduce Coupling" was the tactic category with most relations (although for SOA this number was considerably larger), which emphasizes the importance of loose coupling for systems based on service orientation. Second, since SOA principles had nearly twice the number of mappings than Microservices, the SOA design principles seem to be much more directed towards architectural aspects related to modifiability than the Microservices principles. As already mentioned, the latter span more than architectural aspects, which is why principles like "Products, not Projects" or "Decentralization" are hard to map onto tactics. The principle with the highest number of mappings was "Service Loose Coupling" (7) for SOA while it was "Evolutionary Design" (5) for Microservices. When looking at the tactics with the most mappings to principles in each

[1] https://github.com/xjreb/research-modifiability-tactics.

paradigm, "Maintain Existing Interface" was related to 6 SOA principles and "Restrict Dependencies and Communication Paths" to 3 Microservices principles. The following sections provide more detailed information for each of the principles.

5.1 SOA principles

The SOA principle **Standardized Service Contract** was related to two tactics, namely "Maintain Existing Interface" and "Restrict Dependencies and Communication Paths". Both tactics capture the important SOA concern to restrict how functionality is offered by standardizing service interfaces and maintaining them as long as possible to provide backwards compatibility.

As mentioned above, **Service Loose Coupling** was the SOA principle with most mappings and is unsurprisingly related to each of the 4 tactics from the "Reduce Coupling" category. On top of that, we related it to "Abstract Common Functionality". Although this is primarily a cohesion tactic, the newly created dependency on abstracted and relocated functionality should manifest itself in a loosely coupled way in SOA. Moreover, we related it to "Runtime Registration and Dynamic Lookup" and "Publish-Subscribe", which are defer binding tactics with great importance for SOA that can also help to reduce coupling.

Service Abstraction was mapped onto the coupling tactics "Use Encapsulation", "Maintain Existing Interfaces", and "Use an Intermediary". They reflect the goals of raising the abstraction level for business logic and to hide the implementation of it while providing a stable interface for its access.

The two principles **Service Reusability** and **Service Composability** go hand in hand and have therefore been mapped to the same 4 tactics, namely "Generalize Module", which refers to operations in the service interface in this case, "Abstract Common Functionality", "Use Encapsulation", and "Maintain Existing Interface". These tactics ensure that services are general enough to be reusable by many consumers and provide stability as well as information hiding for sustainable usage.

With 5 mappings, **Service Autonomy** is the SOA principle with the second most relations. The tactics "Use Encapsulation", "Maintain Existing Interface", and "Use an Intermediary" ensure that a service implementation or even its interface can evolve in a consumer-friendly way. "Runtime Registration and Dynamic Lookup" enables a service to autonomously offer its capabilities and to choose from available offerings while "Publish-Subscribe" provides a mechanism to autonomously consume or publish events without having to care too much about potential consumers.

The fairly specialized principle **Service Discoverability** could only be mapped onto one single—albeit perfectly fitting—tactic: "Runtime Registration and Dynamic Lookup".

Lastly, the principle **Service Statelessness** could not be related to any modifiability tactic and is therefore the only SOA principle without a mapping.

5.2 Microservices principles

For the Microservices principles, the total number of mappings was considerably smaller. With 4 relations, **Componentization via Services** was one of the more relevant principles. We mapped it onto the tactics "Abstract Common Functionality", "Use Encapsulation", "Maintain Existing Interface", and "Restrict Dependencies and Communication Paths". These tactics ensure that the only way to access functionality is via the provided interfaces that encapsulate the implementation.

The principle **Organized around Business Capabilities** was related to the "Increase Semantic Coherence" tactic, which perfectly describes the rationale of this principle. In a Microservice-Based System, each service tries to offer only tightly related functionality via the construct of a *Bounded Context*.

For **Smart Endpoints, Dumb Pipes**, the identified relations were rather weak. Nonetheless, we decided to map this principle to the tactics "Restrict Dependencies and Communication Paths" and "Publish-Subscribe", because this principle prescribes limitations for communication channels and a simple topic-based "Publish-Subscribe" mechanism is a potential implementation of a "dumb" pipe.

The **Decentralization** principle was similarly hard to map onto tactics. In the end, we decided to relate it to "Restrict Dependencies and Communication Paths", because it includes the restriction of not sharing databases. Data owned by a service has to be accessed via the API of this service.

As expected, the **Evolutionary Design** principle maps rather well onto modifiability tactics and is the Microservices principle with the most relations (5). The cohesion tactics "Increase Semantic Coherence", "Anticipate Expected Changes", and "Split Module" perfectly describe the service decomposition and refactoring philosophy of Microservices. Furthermore, "Use Encapsulation" and "Maintain Existing Interface" add to this by providing important foundations for sustainable evolution.

For **Products, not Projects**, there was no tactic that could be related, since it describes organizational aspects. Likewise, the **Design for Failure** principle is related to reliability and could probably be mapped very well onto availability tactics. Since we are only concerned with modifiability in this paper, this Microservices principle is the second one with zero relations.

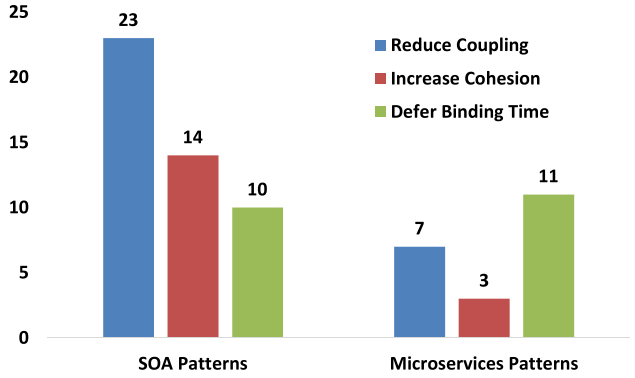

Fig. 3 Number of related patterns per tactic category

5.3 SOA and Microservices patterns

As the second mapping activity, we analyzed the relations of 118 SOA patterns and 42 Microservices patterns with modifiability tactics. Because it was sometimes hard to choose a single tactic for a pattern and we did not want to relate one pattern to several tactics, we compressed multiple closely related tactics together in some cases (e.g. "Abstract Common Functionality" and "Generalize Module"). For SOA, we were able to relate a total of 47 patterns (~ 40%) while for Microservices we mapped a total of 21 patterns (50%). The "Reduce Coupling" tactic category was again the most prominent category for SOA with 23 patterns (~ 49%), but for Microservices it was "Defer Binding Time" with 11 patterns (~ 52%). Figure 3 presents more details about the category distribution. The following sections provide more information for each tactic or tactics collection. For the complete list of patterns, please refer to the GitHub repository.[2]

Since the three tactics **Increase Semantic Coherence**, **Anticipate Expected Changes**, and **Split Module** can be realized by roughly the same patterns, they were collectively addressed. In a service-oriented context, these tactics are mostly concerned with decomposition and normalization. 6 SOA patterns have been related to this incentive, e.g. "Service Decomposition", "Service Refactoring", or "Lightweight Endpoint". Two Microservices patterns concerned with decomposition have been chosen as well, namely "Decompose by Business Domain" and "Decompose by Subdomain".

Also related to cohesion, the two tactics **Abstract Common Functionality** and **Generalize Module** were treated in unison. Since generalizing capabilities to create agnostic and therefore reusable functionality is an important theme in SOA, 8 patterns were identified for these tactics, e.g. "Entity Abstraction", "Canonical Resource", or "Service Host". For Microservices, one pattern was chosen, namely "Microservice Chassis", which is the equivalent of "Service Host".

Coupling tactics were sufficiently distinct to not require compression. The **Use Encapsulation** tactic can be realized by the basic SOA patterns "Service Encapsulation" and "Decoupled Contract" while in a Microservices context "Service-per-Container" or "Service-per-VM" can be used. For **Maintain Existing Interface**, SOA provides the patterns "Canonical Versioning", "Compatible Change", and "Version Identification". A specific Microservices pattern could not be identified for this tactic, although the versioning of e.g. RESTful APIs is also a popular theme in this space.

With governance and centralization being important SOA properties, the tactic **Restrict Dependencies and Communication Paths** can be realized by 8 patterns, e.g. "Canonical Protocol", "Canonical Schema", or "Official Endpoint". Similarly, 2 Microservices patterns have been identified, "Database per Service" and "Command Query Responsibility Segregation (CQRS)".

Use an Intermediary with the goal to reduce coupling by indirection was the tactic with the most SOA pattern matches (9). It can be realized by e.g. "Service Façade", "Proxy Capability", or "Service Broker". For Microservices, we chose 3 patterns, "API Composition", "API Gateway", and "Backend for Front End".

Tactics that defer bindings could also be treated on a single basis. The tactic **Runtime Registration and Dynamic Lookup** which is so central to service orientation matches surprisingly only one SOA pattern, "Metadata Centralization". On the Microservices side however, we identified 5 patterns, e.g. "Self Registration", "Client-side Discovery", or "Server-side Discovery", which makes this the tactic with the most Microservices pattern matches.

For **Runtime Binding**, it is the other way around. While 6 SOA patterns were chosen (e.g. "Intermediate Routing", "Workflodize", or "Endpoint Redirection"), not a single Microservices pattern could be related to this tactic.

The **Publish-Subscribe** tactic can be related to 3 patterns in each context, namely "Service Messaging", "Event-Driven Messaging", and "Inversion of Communication" for SOA, while "Event Sourcing", "Application Events", and "Messaging" were identified for Microservices.

Lastly, **Start-Up Time Binding** had no related SOA pattern, but can be realized by the Microservices pattern "Externalized Configuration". Similarly, no SOA pattern could be identified for **Deployment Time Binding** while 2 Microservices patterns are available, "Service Deployment Platform" and "Serverless Deployment". For the **Compile Time Binding** tactic, no pattern could be related in both of the service-based styles, which makes this the only modifiability tactic without pattern mappings.

[2] https://github.com/xjreb/research-modifiability-tactics.

6 Discussion (RQ3)

When trying to interpret the mapping results several things are of interest. First, the SOA principles are of a more general and technical nature and therefore produced a larger number of mappings to architectural modifiability than the Microservices principles (26 vs. 15). Interestingly, SOA focuses on a smaller number of tactics (7 of 15 tactics had no mapping, the coupling category had more than 60% of mappings) while the latter spread more evenly (only 5 tactics with no mapping, coupling category with ~ 47%). The Microservices principles often fit extremely well for just a few tactics, e.g. the principle "Organized around Business Capabilities" and the tactic "Increase Semantic Coherence". If we had specified the strength of each mapping (which is very difficult to achieve in an objective way), the results might not look as far apart as is currently suggested. Furthermore, the "Decentralization" principle could only be mapped to one tactic, but could potentially influence modifiability further via the development process and technology choices, both of which are not really represented amongst the existing tactics.

Second, when looking at the mappings of patterns to tactics, we could relate ~ 10% more Microservices patterns than SOA patterns, although both the total number as well as the number of mapped patterns was smaller (SOA: 47 of 118, Microservices: 21 of 42). This could suggest that Microservices are slightly more tailored towards modifiability in this regard. During the analysis, however, we identified a number of SOA patterns that we instinctively deemed beneficial for modifiability but that we could not easily relate to existing tactics, e.g. "Process Abstraction", "Process Centralization", or "Brokered Authentication". When taking this into consideration, the total degree of pattern mappings of the two service-based styles seems to be similarly close as in the case of the principles. The distribution, however, is focused on different categories. For SOA, ~ 49% of the mappings are in the category "Reduce Coupling" while for Microservices, ~ 52% of mapped patterns are in "Defer Binding Time". Interestingly, only 3 of the 21 mapped Microservices patterns are in the "Increase Cohesion" category, although small and highly cohesive services according to *Bounded Contexts* are a key philosophy within Microservice-Based Systems. This suggests a lack of systematic approaches and detailed patterns to achieve this postulated property.

Unsurprisingly, mappings around loose coupling were very prevalent throughout both service-based styles which quantitatively strengthens the importance of this characteristic. The "Reduce Coupling" tactic category had most mappings for both principle lists (SOA: 16, Microservices: 7), the most mappings for the SOA patterns (23) and the second most for the Microservices patterns (7). Moreover, "Service Loose Coupling" was the most frequently mapped principle (7). A rich variety of patterns exists to ensure this important service-oriented design property.

When comparing the two styles further, however, they both strive for modifiability in different ways. For SOA, governance through restrictions, reusability via abstraction of common functionality, and the usage of intermediaries are very important. Furthermore, interoperability, integration, and preserving interfaces are common themes. Microservices on the other hand focus more on "Evolutionary Design" as a key principle to achieve modifiability. They also bring in a focus on "Infrastructure Automation" which can be related to modifiability tactics (deployment and compile time binding), although it is not really an architectural property and could also be applied to SOA. Detailed patterns for "Runtime Registration and Dynamic Lookup" further strengthen the importance of the "Defer Binding Time" category for Microservices. Interestingly, "Restrict Dependencies and Communication Paths" was the tactic with most principle mappings (3) which at first seems counterintuitive to the decentralized and heterogeneous nature of Microservices. When examining these mappings, however, the restrictions are based on the usage of service interfaces as the sole access point for functionality, the restriction to not share databases, and the incentive to keep as much logic as possible inside services and not inside communication channels. So, while Microservices generally are not related to strong governance, they provide standardization on a few foundational topics and achieve a base level of modifiability with this. In a similar fashion Microservices concentrate on just a small number of communication protocols (most often RESTful HTTP and a single asynchronous messaging protocol like AMQP) while SOA allows for a much higher degree of heterogeneous interoperability, usually via a central Enterprise Service Bus (ESB).

Several threats to validity need to be mentioned with this research. First, the qualitative nature of the approach ties the results to the personal experience and judgment of the authors. This introduces the possibility of subjective bias and could limit research reproducibility. Moreover, Microservices are much younger than SOA and can therefore not provide such an established foundation of principles and patterns. This could add an uneven and temporary connotation to the comparison. As already mentioned, several principles and patterns are applicable in both architectural styles or even exist under a different name. Lastly, our results scrutinize "theoretical" modifiability. In a practical setting, software developers can create systems of arbitrary evolvability in both architectural styles. The actual value of this research relies on how well the results can be transferred to a practical context. In its current form,

practitioners may have difficulties to leverage this research for design and implementation activities.

7 Summary and conclusion

To strengthen systematic knowledge of service-oriented evolution qualities, we related 15 architectural modifiability tactics in 3 categories to a) principles for both SOA and Microservices and b) design patterns for SOA and Microservices. As an answer for RQ1, we identified 26 mappings for SOA principles while the mapping number for Microservices principles was 15. In the case of patterns (RQ2), we could map ~ 40% of SOA and 50% of Microservices pattern candidates to tactics. While the two service-based styles showed some similarities in how they achieve modifiability (focus on loose coupling and encapsulation), we could also extract notable differences (e.g. standardization and governance vs. evolutionary design) that seem to be in line with the general understanding of these types of systems.

As a result of these mapping activities and to provide a partial answer to RQ3, we can conclude that both service-based styles have several intrinsic properties that are beneficial for modifiability and that a wide variety of patterns exists for concrete realizations of the presented tactics. The main difficulty, however, will be to summarize the essence and preconditions of these qualities in an easily consumable form for software professionals. Follow-up research will be concerned with methods to use these theoretical results in an unobtrusive and efficient way for practical design and implementation scenarios.

Acknowledgements 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. Alexander C, Ishikawa S, Silverstein M, i Ramió JR, Jacobson M, Fiksdahl-King I (1977) A pattern language. Gustavo Gili
2. Ali M, Elish MO (2013) A comparative literature survey of design patterns impact on software quality. In: 2013 international conference on information science and applications (ICISA), pp 1–7. https://doi.org/10.1109/ICISA.2013.6579460
3. Ameller D, Galster M, Avgeriou P, Franch X (2016) A survey on quality attributes in service-based systems. Softw. Qual. J. 24(2):271–299. https://doi.org/10.1007/s11219-015-9268-4
4. Bachmann F, Bass L, Nord R (2007) Modifiability tactics. Tech. Rep. CMU/SEI-2007-TR-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA
5. Baker S, Dobson S (2005) 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. https://doi.org/10.1007/11575771_40
6. Bass L, Clements P, Kazman R (2003) Software architecture in practice, 2nd edn. Addison-Wesley Professional, Westford
7. Bass L, Clements P, Kazman R (2012) Software architecture in practice, 3rd edn. Addison-Wesley Professional, Westford
8. Bogner J, Fritzsch J, Wagner S, Zimmermann A (2018) Limiting technical debt with maintainability assurance - an industry survey on used techniques and differences with Service- and Microservice-Based Systems. In: Proceedings of the 1st international conference on technical debt (TechDebt'18). ACM, Gothenburg, Sweden. https://doi.org/10.1145/3194164.3194166
9. Bogner J, Zimmermann A, Wagner S (2018) Analyzing the relevance of SOA patterns for Microservice-Based Systems. In: Proceedings of the 10th central European workshop on services and their composition (ZEUS'18). CEUR-WS.org
10. Capelli S, Scandurra P (2016) A framework for early design and prototyping of service-oriented applications with design patterns. Comput Lang Syst Struct 46:140–166. https://doi.org/10.1016/j.cl.2016.07.001
11. Erl T (2005) Service-Oriented Architecture: concepts, technology, and design. Prentice Hall PTR, Upper Saddle River
12. Erl T (2009) SOA design patterns. Pearson Education, Boston
13. Erl T, Carlyle B, Pautasso C, Balasubramanian R (2012) SOA with REST: principles, patterns and constraints for building enterprise solutions with REST. The prentice hall service technology series from Thomas Erl. Pearson Education
14. Fowler M (2015) Microservices resource guide. URL http://martinfowler.com/microservices
15. Galster M, Avgeriou P (2012) Qualitative analysis of the impact of SOA patterns on quality attributes. In: 2012 12th international conference on quality software, pp 167–170. https://doi.org/10.1109/QSIC.2012.35
16. Gamma E, Helm R, Johnson R, Vlissides J (1994) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Boston
17. Hegedűs P, Bán D, Ferenc R, Gyimóthy T (2012) Myth or reality? Analyzing the effect of design patterns on software maintainability. In: Kim Th, Ramos C, Kim Hk, Kiumi A, Mohammed S, Ślęzak D (eds) communications in computer and information science, communications in computer and information science, vol 340, pp. 138–145. Springer, Berlin. https://doi.org/10.1007/978-3-642-35267-6_18
18. Mcheick H, Qi Y (2012) Quality attributes and design decisions in service-oriented computing. In: 2012 international conference on innovations in information technology (IIT), IEEE, pp 283–287. https://doi.org/10.1109/INNOVATIONS.2012.6207749
19. Me G, Procaccianti G, Lago P (2017) Challenges on the relationship between architectural patterns and quality attributes. In: 2017 IEEE international conference on software architecture (ICSA), IEEE, pp 141–144. https://doi.org/10.1109/ICSA.2017.19
20. Mirandola R, Potena P, Scandurra P (2014) Adaptation space exploration for service-oriented applications. Sci Comput Program 80(PART B):356–384. https://doi.org/10.1016/j.scico.2013.09.017
21. Newman S (2015) Building microservices: designing fine-grained systems, 1st edn. O'Reilly Media, Newton
22. Papazoglou MP (2003) Service-oriented computing: concepts, characteristics and directions. In: Proceedings of the 7th international conference on properties and applications of dielectric materials (Cat. No.03CH37417), IEEE Comput. Soc., pp 3–12. https://doi.org/10.1109/WISE.2003.1254461
23. Parvizi-Mosaed A, Moaven S, Habibi J, Beigi G, Naser-Shariat M (2015) Towards a self-adaptive service-oriented methodology based on extended SOMA. Front Inf Technol Electron Eng 16(1):43–69. https://doi.org/10.1631/FITEE.1400040

24. Parvizi-Mosaed A, Moaven S, Habibi J, Heydarnoori A (2014) Towards a tactic-based evaluation of self-adaptive software architecture availability. In: The 26th international conference on software engineering and knowledge engineering, hyatt regency, Vancouver, BC, Canada, July 1–3, 2013, pp 168–173
25. Perepletchikov M, Ryan C, Frampton K (2005) 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, pp 431–441. https://doi.org/10.1007/11575863_63
26. Riaz M, Breaux T, Williams L (2015) How have we evaluated software pattern application? A systematic mapping study of research design practices. Inf Softw Technol 65:14–38. https://doi.org/10.1016/j.infsof.2015.04.002
27. Richards M (2016) Microservices vs. Service-Oriented Architecture. O'Reilly Media, Sebastopol
28. Richardson C (2018) Microservices patterns. Manning Publications, Shelter Island
29. Rotem-Gal-Oz A (2012) SOA patterns. Manning, Shelter Island
30. Voelz D, Goeb A (2010) What is different in quality management for SOA? In: 2010 14th IEEE international enterprise distributed object computing conference, IEEE, pp 47–56. https://doi.org/10.1109/EDOC.2010.27
31. Yu Y, Fernandez-Ramil J, Lu J, Yuan P (2007) Comparing web services with other software components. In: Proceedings—2007 IEEE international conference on web services, ICWS 2007, Icws, IEEE, pp 388–397. https://doi.org/10.1109/ICWS.2007.64
32. Zimmermann O (2017) Microservices tenets. Comput Sci Res Dev 32(3–4):301–310. https://doi.org/10.1007/s00450-016-0337-0

Justus Bogner holds a B.Sc. in Applied Computer Science from Baden-Wuerttemberg Cooperative State University Stuttgart, Germany, and a M.Sc. in Services Computing from the University of Applied Sciences Reutlingen, Germany. He is currently a PhD student in the cooperative doctoral program "Services Computing" at the Herman Hollerith Center (HHC) in Boeblingen, a joint initiative between the University of Stuttgart and the University of Applied Sciences Reutlingen. His research interests are in the area of software engineering and software architecture for long-living Service- and Microservice-Based Systems, specifically their sustainable maintenance and evolution.

Stefan Wagner studied computer science in Augsburg and Edinburgh. He holds a PhD from the Technical University of Munich. He is the author of more than 120 peer-reviewed scientific articles and the book "Software Product Quality Control". He is a member of the editorial boards of MDPI Computer and Research Ideas and Outcomes. His research interests include various aspects of software and systems engineering including software quality, requirements engineering, safty, security and privacy, as well as agile software development. He is a member of ACM, IEEE, and the German GI.

Alfred Zimmermann is a Professor of Computer Science in the specialty Digital Enterprise Architecture at Reutlingen University, Germany. He is Director of Research and Speaker of the Doctoral Program for Services Computing at the Herman Hollerith Center Boeblingen, Germany. His research is focused on digital transformation and digital enterprise architecture with decision analytics in close relationship with digital strategy and governance. He graduated in Medical Informatics at the University of Heidelberg, Germany and obtained his PhD in Informatics from the University of Stuttgart, Germany. Beside his academic experience, he has a strong practical background as a technology manager and leading consultant at Daimler AG, Germany. Prof. Zimmermann keeps academic relations of his home university to the German Computer Science Society (GI), the ACM, and the IEEE. He serves in different editorial boards and program committees and publishes results from his research at conferences, workshops, as well as in books and journals.