Exploring Maintainability Assurance Research for Service- and Microservice-Based Systems: Directions and Differences

Justus Bogner
University of Applied Sciences Reutlingen, Herman Hollerith Center, Germany
University of Stuttgart, Institute of Software Technology, Software Engineering Group, Germany
https://www.hhz.de/en/research/research-groups/digital-enterprise-architecture
justus.bogner@iste.uni-stuttgart.de

Adrian Weller
University of Stuttgart, Institute of Software Technology, Software Engineering Group, Germany
https://www.iste.uni-stuttgart.de/se
adrian.weller94@gmail.com

Stefan Wagner
University of Stuttgart, Institute of Software Technology, Software Engineering Group, Germany
https://www.iste.uni-stuttgart.de/se
stefan.wagner@iste.uni-stuttgart.de

Alfred Zimmermann
University of Applied Sciences Reutlingen, Herman Hollerith Center, Germany
https://www.hhz.de/en/research/research-groups/digital-enterprise-architecture
alfred.zimmermann@reutlingen-university.de

Abstract

To ensure sustainable software maintenance and evolution, a diverse set of activities and concepts like metrics, change impact analysis, or antipattern detection can be used. Special maintainability assurance techniques have been proposed for service- and microservice-based systems, but it is difficult to get a comprehensive overview of this publication landscape. We therefore conducted a systematic literature review (SLR) to collect and categorize maintainability assurance approaches for service-oriented architecture (SOA) and microservices. Our search strategy led to the selection of 223 primary studies from 2007 to 2018 which we categorized with a threefold taxonomy: a) architectural (SOA, microservices, both), b) methodical (method or contribution of the study), and c) thematic (maintainability assurance subfield). We discuss the distribution among these categories and present different research directions as well as exemplary studies per thematic category. The primary finding of our SLR is that, while very few approaches have been suggested for microservices so far (24 of 223, ~11%), we identified several thematic categories where existing SOA techniques could be adapted for the maintainability assurance of microservices.

2012 ACM Subject Classification Software and its engineering → Software evolution; Software and its engineering → Maintaining software; Social and professional topics → Quality assurance; Information systems → Web services

Keywords and phrases Maintainability, Software Evolution, Quality Assurance, Service-Based Systems, SOA, Microservices, Systematic Literature Review


Supplement Material https://github.com/xJREB/slr-maintainability-assurance

Funding Justus Bogner: Funded by the Ministry of Science of Baden-Württemberg, Germany, for the doctoral program Services Computing (https://www.services-computing.de/?lang=en).
Introduction

While software continues “to eat the world” [3], it becomes all the more important that systems can be quickly adapted to new or changed requirements. As more and more business processes are not only supported by software, but digital goods and services form the essence of entire businesses, the sustainable evolution of the underlying software is a vital concern for many enterprises. The associated quality attribute is referred to as maintainability [35], i.e. the degree of effectiveness and efficiency with which a system can be modified. In fast-moving markets, the adaptive and extending notion of this quality attribute – also referred to as evolvability [68] – is especially crucial. To address this, software professionals rely on a set of diverse activities aiming to ensure a sufficient degree of maintainability. We refer to these approaches and techniques as maintainability assurance. In general, such activities are either of an analytical nature, i.e. to identify existing issues, or of a constructive nature, i.e. to remediate or prevent issues [79]. Examples for analytical techniques include metric-based evaluation (both static and dynamic analysis), scenario-based evaluation, or code review. Examples for constructive techniques are refactoring, standardization, or systematic maintainability construction with design patterns. Furthermore, maintainability assurance for larger systems is often structured into a communicated assurance process and is an integral part of the development flow.

The introduction of service-oriented computing [59] arguably led to several maintainability-related benefits such as encapsulation, strict separation between interface and implementation, loose coupling, composition, and reuse. The two service-based architectural styles – namely service-oriented architecture (SOA) [23] and microservices [53] – are very popular for implementing enterprise applications or web-based systems with strong requirements for scalability and maintainability. The younger microservices paradigm also places special emphasis on evolutionary design [29]. Several publications have tried to summarize the differences and commonalities between SOA and microservices [66, 89, 21, 12]. While no holistic consensus has been reached so far (and probably never will be), many authors focus on the broad set of architectural commonalities. Highlighted differences of microservices are e.g. their decentralized governance and organization (as opposed to centralization and standardization in SOA), their focus on very few lightweight communication protocols like RESTful HTTP (as opposed to protocol-agnostic interoperability via an enterprise service bus in SOA), or their “share nothing” principle (as opposed to SOA’s focus on abstraction and reuse). Nonetheless, the majority of publications acknowledges the shared service-oriented principles like loose coupling, location transparency, or statelessness. Early adopters from industry like Netflix also referred to their system as “fine-grained Service Oriented Architecture” [80].

In principal, both SOA and microservices are based on beneficial properties for maintainable and evolvable systems. However, concrete maintainability assurance processes for such systems are still not trivial to establish. Empirical studies about industry practices in this regard also highlighted that there is a high trust in the base maintainability of service orientation, which may even lead developers to actively reduce assurance efforts [78, 9]. Simultaneously, practitioners are uncertain how to handle service-oriented particularities in this regard and especially report challenges for architectural evolvability [9, 10]. When trying to get an overview of assurance approaches for service orientation proposed by academia, it is not easy to quickly scan the scattered variety of existing publications. Researchers have suggested a plethora of assurance techniques specifically designed for SOA, microservices, or both that try to approach maintainability from different directions. To enable such an overview, we therefore conducted a systematic literature review (SLR) to collect and
categorize existing maintainability assurance research for service orientation. Our review explicitly included both SOA and microservices, because the number of relevant publications for the younger architectural style would have been too small on its own. Moreover, we believed that a lot of the approaches designed for SOA will have relevance for microservices as well. Lastly, the inclusion of both service-based styles will enable an additional comparison on this level. Our SLR was guided by the following four research questions:

- **RQ1**: How can maintainability assurance research proposed for service- and microservice-based systems be categorized?
- **RQ2**: How are the identified publications distributed among the formed categories?
- **RQ3**: What are the most relevant research directions per identified category?
- **RQ4**: What are notable differences between the approaches proposed for service-based systems and those for microservices?

In the remainder of this paper, we first present related literature studies (Section 2) and describe the details of our own study design (Section 3). After that, we present the SLR results, from which we synthesized the answers to our research questions (Section 4). Lastly, we mention possible threats to validity (Section 5) and conclude with a summary as well as an outlook on potential follow-up research (Section 6).

## 2 Related Work

Several existing literature studies cover maintainability-related aspects without focusing on a specific system type or architectural style. Breivold et al. [14] conducted an SLR to collect studies on the architectural analysis and improvement of software evolvability. They identified 82 primary studies that they structured into five categories, such as quality considerations during design, architectural quality evaluation, architectural knowledge management, or modeling techniques. Service-oriented approaches are not included.

Similarly, Venters et al. [77] synthesized existing research approaches for general architecture sustainability in a non-systematic review. They define sustainability as a system’s capacity to endure. Service-oriented approaches are only briefly mentioned in the area of reference architectures, where some proposals for SOA are listed. Other described topics are the importance of architectural decisions or metrics for the quantification of sustainability.

There are also several service-based literature studies focusing on SOA. Back in 2009, Gu and Lago [33] conducted an SLR to uncover general service-oriented system challenges. Using 51 primary studies, they identified more than 400 challenges, most of them related to quality attributes, service design, and data management. Only three reported challenges were associated with maintenance, i.e. their review is broader than our intended scope.

A more fine-grained scope than the one intended by us was applied in the literature review of Bani-Ismail and Baghdadi [6]: they solely focused on service identification (SI) in SOA and derived eight different challenges for this activity. The maintainability-related aspect of service granularity is presented as one of the most important challenges.

In another service-oriented SLR, Sabir et al. [69] analyzed the evolution of object-oriented and service-oriented bad smells as well as differences with their detection mechanisms. From 78 publications, they identified 56 object-oriented and 19 service-based smells and presented details about their detection approaches. Smells related to microservices are not covered.

In similar fashion, Bogner et al. [8] conducted an SLR to collect existing antipatterns for both SOA and microservices. While they did not include many details on detection approaches, they synthesized a holistic data model for all antipatterns and also created a taxonomy for their categorization. 14 of the 36 antipatterns were categorized as applicable to SOA, three to microservices, and 19 to both styles. Like the review of Sabir et al., this is a subset of our intended scope, but nonetheless targets both SOA and microservices.
Several more recent literature studies also focus exclusively on microservices. Di Francesco et al. [20] used a systematic mapping study to create a research overview on architecting with microservices. They derived a classification framework and used it to produce a systematic map of the topics of 103 selected primary studies. While maintainability is mentioned in 43 studies as an important design goal or investigated quality attribute, the broad scope of the review prevents a more detailed discussion of how maintainability is actually ensured.

Lastly, Soldani et al. [73] systematically surveyed the existing grey literature on microservices to distill their technical and operational “pains and gains”. Afterwards, identified concerns were assigned to common stages of the software life cycle such as design or operation. Maintainability is briefly discussed as an advantageous “gain” based on small service size and self-containment, but concrete techniques for its assurance are not mentioned.

In summary, none of the presented related studies focus exclusively on maintainability and its assurance while simultaneously targeting service- and microservice-based systems. The studies that have a similar scope do not limit their review to service orientation and the ones that do are either too general or too specific in their discussed aspects. We intend to close this gap by presenting an SLR that specifically analyzes the state of the art of maintainability assurance for service- and microservice-based systems.

3 Research Methodology

In general, an SLR is a secondary study that is used to identify, analyze, and summarize (scientific) publications within a certain research area of interest. As such, it presents an overview of the state of the art in a certain (sub)field and may point out research gaps or even a research agenda to close them. Since scientific rigor and replicability are very important in such studies, we relied on the process and guidelines described by Kitchenham and Charters [39]. Moreover, we published all research artifacts in an online repository.

Our general research process for this study was as follows (see also Fig. 1). First, we brainstormed about research questions we intended to answer and defined four questions that built upon each other (see Section 1). As a second step, we designed a detailed protocol to guide us through the review. This protocol contained the used data sources (literature databases and search engines), a search term with keywords, filter criteria for inclusion and exclusion of studies, as well as a description of the process. We then used this protocol to retrieve the initial result set from all data sources and subsequently applied our filter criteria. The first two authors individually analyzed and filtered all identified publications and afterwards compared the results. Any differences were discussed until a consensus was reached. For the remaining studies, we performed one round of forward citation search (“snowballing”) and applied the same filter criteria to the newly identified publications (again with two researchers). Included studies were merged into the existing set and duplicates were removed. This final set of primary studies was now analyzed in an iterative process. A categorization scheme was derived and subsequently applied to the publications. The result was then analyzed again and possible improvements for the categorization scheme were implemented, which led to the next round of categorization. As with the inclusion and exclusion criteria, categorization of the whole set was performed by two researchers, who discussed any difference of opinion. Once the categories were stable, we started the detailed evaluation to synthesize the answers to our research questions.

1 https://github.com/xJREB/slr-maintainability-assurance
Define research questions
Design SLR protocol
Retrieve initial search results
Apply filter criteria
Perform forward citation search

Analyze results
Merge results
Apply filter criteria
Perform detailed evaluation
Synthesize answers to RQs

Derive categorization scheme
Categorize results
[ if categories are stable ]
Perform detailed evaluation
Synthesize answers to RQs

Figure 1 General Research Process.

The four used data sources for our initial search (see also Fig. 2) were IEEE Xplore, ACM Digital Library, Springer Link, and ScienceDirect, as they are very common for software engineering and service-oriented topics. For the snowballing phase, we relied on the publisher-agnostic search engine Google Scholar.

- IEEE Xplore: https://ieeexplore.ieee.org
- ACM Digital Library: https://dl.acm.org
- Springer Link: https://link.springer.com
- ScienceDirect (Elsevier): https://www.sciencedirect.com
- Google Scholar (only for snowballing): https://scholar.google.com

Figure 2 Used Digital Libraries and Search Engines for the SLR.

As our search string (see also Fig. 3), we formed two buckets with keywords. The two buckets were combined with an AND relation while the keywords within each bucket were combined with an OR relation, i.e. from each bucket, at least one term needed to match. The first bucket contained our central quality attribute maintainability as well as the closely related terms modifiability, evolvability, and evolution. The second bucket was responsible for our targeted system types and therefore consisted of the terms soa, microservice, service-oriented, and service-based. The search string was not confined to any particular field.

(maintainability ∨ modifiability ∨ evolvability ∨ evolution) ∧ (soa ∨ microservice ∨ service-oriented ∨ service-based)

Figure 3 Used Search String for the SLR.

Since we only relied on manual filtering to avoid the accidental exclusion of fitting studies that just use different keywords, we also had to limit the result set to a manageable amount. We therefore only considered the first 250 results per data source, i.e. we had a total of 1000 publications for manual analysis. As our most basic inclusion criteria, we only considered publications written in English and published in the years 2007 up until 2018.
The title and abstract of studies passing this test were then assessed for their relevance to our research questions. The main focus of the paper needed to be on analyzing or improving maintainability (or a related quality attribute or design property) in the context of service-oriented computing (e.g. SOA or microservices). For example, the architecture sustainability review of Venters et al. [77] fulfills the first property, but is not primarily about service orientation. Conversely, the SOA policy optimization approach from Inzinger et al. [36] is clearly about service orientation, but does not solely target maintainability. If the topic relevance could not be determined from title and abstract alone, other parts of the paper like the introduction or conclusion had to be read. Finally, we excluded the fields of runtime adaptation as in [26], software testing as in [37], and legacy to SOA or microservices migration as in [47]. While these topics are related to maintenance and evolution, they are very specialized and each one could probably provide enough material for a separate SLR.

4 Results

Using the process described above, we obtained an initial set of 1000 papers, i.e. 250 per selected publisher. We then applied our inclusion criteria, which resulted in a filtered set of 122 papers. In the snowballing phase, we identified a total of 806 publications that cited a paper from our filtered set. Lastly, we applied the same filter criteria to these new publications and merged the remaining ones back into the filtered set while removing duplicates. This resulted in a final set of 223 primary studies (see also Fig. 4). Nearly half of these papers (105) were published by IEEE, followed by 40 Springer publications (18%), 31 papers from ACM (14%), and 10 from ScienceDirect (4%). The remaining 37 publications (17%) were from 31 different publishers (see also Fig. 5). When looking at the number of publications per year (Fig. 6), we see a slow beginning in 2007 (six publications), a peak in 2011 (35 publications), smaller yet fairly similar numbers for 2012 to 2015 (22-24 publications), and finally another decline for 2016 to 2018 (15-18 publications).

![Figure 4 SLR Results: Number of Publications per Stage.](image)

4.1 Research Categories (RQ1)

To answer the first research question, we derived a three-dimensional taxonomy to categorize the identified primary studies (see Table 1). The first and most obvious category type called architectural consisted of three different categories that determined if a study targeted SOA, microservices, or both architectural styles. Both was selected if the study either explicitly stated the inclusion of both SOA and microservices or if it was about concepts like RESTful services that are prevalent in both styles. This category type was mandatory and exactly one
Figure 5 Publisher Distribution. Figure 6 Number of Publications per Year.

Table 1 Three-Dimensional Categorization Scheme.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Mandatory</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural</td>
<td>Contains three categories that determine if a publication focuses on SOA, microservices, or both styles.</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Methodical</td>
<td>Contains five categories that either determine the used research method (e.g. literature study) or the study’s contribution (e.g. model or taxonomy).</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Thematic</td>
<td>Contains nine categories that determine the topic of a publication, i.e. subfields of maintainability assurance.</td>
<td>Yes</td>
<td>Yes (except for other)</td>
</tr>
</tbody>
</table>

category had to be chosen per publication. Second, the optional methodical category type determined either the applied methodology (e.g. case study) or the provided contribution (e.g. process or method). It consisted of five different categories, from which multiple ones could be selected per study. Lastly, the most important thematic category type determined the actual maintainability-related topic of a publication, i.e. a more specific subfield of maintainability assurance. To avoid a large number of very fine-grained thematic categories, we created the generic Other category. At least one thematic category had to be chosen per publication. The following listing briefly describes all methodical categories.

- **Case, Field, or Empirical Study**: the publication either describes a case study (e.g. demonstrating an approach with an example system), a field study (e.g. analyzing an industry system), or an empirical study (e.g. a survey, interviews, or a controlled experiment)
- **Literature Study**: the publication presents the results of a literature study like an SLR or a systematic mapping study
- **Model or Taxonomy**: the publication contributes a (meta) model or taxonomy to further the conceptual understanding of a topic
- **Process or Method**: the publication defines a method or process, i.e. a sequence of activities to achieve a certain goal
- **Reference Architecture or Tool**: the publication either describes a reference architecture (an abstract and reusable template to create system architectures) or a tool to e.g. mitigate manual efforts

Lastly, the next listing presents all thematic categories. They refer exclusively to a service-based context.
- **Architecture Recovery and Documentation**: relying on architecture reconstruction (if no current documentation is available) or on general architecture documentation support to increase analyzability and therefore maintainability; example: [40]
- **Model-Driven Approaches**: approaches that rely on model-driven engineering to reduce long-term maintenance efforts with e.g. code generation from machine-readable models; example: [49]
- **Patterns**: applying patterns specifically designed for service orientation to systematically improve maintainability-related aspects or to describe service evolution; example: [83]
- **Antipatterns and Bad Smells**: conceptualizing service-based antipatterns and bad smells or providing detection approaches for them to identify maintainability weaknesses; example: [58]
- **Service Identification and Decomposition**: approaches to identify suitable service boundaries for functionality or to decompose large existing services into more fine-grained ones that are more beneficial for maintainability; example: [45]
- **Maintainability Metrics and Prediction**: conceptualizing or evaluating service-based metrics to analyze or predict maintainability; example: [50]
- **Change Impact and Scenarios**: approaches for analyzing the potential propagation of service changes or general scenario-based maintainability evaluation; example: [34]
- **Evolution Management**: general approaches to support or improve the overall service evolution process via e.g. systematic planning techniques, accelerating the process, increasing fault tolerance, or mitigating other negative consequences; example: [28]
- **Other**: all papers that could not be assigned to one of the other categories were sorted into this one; example: [84]

### 4.2 Category Distributions (RQ2)

The analysis of distributions among architectural categories (SOA, microservices, both) immediately revealed that nearly 90% of the 223 publications exclusively targeted SOA (see Fig. 7). Only 12 publications solely referred to microservices while an additional 12 included both SOA and microservices. This means that only a combined ~11% of our primary studies were about maintainability assurance approaches for microservices. Since microservices are much younger than SOA, it also makes sense to look at a more recent subset, e.g. starting from 2014 when microservices began to rise in popularity (see Fig. 8). However, from 2014 to 2018, identified microservices-related publications still only accounted for a combined ~25%.

![Figure 7 Architectural Categories 2007-2018.](image1)

![Figure 8 Architectural Categories 2014-2018.](image2)
Concerning *methodical* categories (see Fig. 9), the most frequent one was *case, field, or empirical study*: 63% of publications included such a study, most often to demonstrate or evaluate a proposed approach or to analyze industry practices. Moreover, nearly half of the publications (110) described a *process or method* as part of their contribution, which highlights the importance of systematic approaches in this field. Both the conceptual contribution of a *model or taxonomy* (75 of 223) and the more practical contribution of a *reference architecture or tool* (72 of 223) were present in roughly one third of studies. Lastly, 25 publications (11%) described a *literature study* for meta analysis. Overall, 124 publications were associated with at least two methodical categories (56%) and 68 publications with at least three (30%).

With respect to *thematic categories* (Fig. 10), around half of the publications were equally distributed among either *evolution management* (56) or *maintainability metrics & prediction* (55). This shows the popularity of approaches to systematically manage service evolution and mitigate potential consequences as well as the interest in maintainability metrics specifically designed for service orientation. In the remaining half, the largest category accounting for 17% of total publications was *change impact & scenarios*, which indicates that qualitative evaluation has not been as popular as quantitative metric-based evaluation so far. Smaller categories were *antipatterns & bad smells* (9%), *service identification & decomposition* (9%), *patterns* (6%), *model-driven approaches* (5%), and *architecture recovery & documentation* (4%). Lastly, 18 publications were categorized with *other* (8%) because they did not fit into any existing category. As opposed to methodical categories, multiple selection was quite rare here, i.e. only 21 publications were related to more than one category (9%).

---

**Figure 9** Distribution: Methodical Categories (percentages are relative to 223 publications).

**Figure 10** Distribution: Thematic Categories (percentages are relative to 223 publications).
4.3 Research Directions per Category (RQ3)

In this section, we briefly describe the most relevant research directions per identified thematic category (except for other). We illustrate these by describing selected exemplary studies.

**Architecture Recovery and Documentation.** In our smallest thematic category, the majority of the nine publications was concerned with (semi-)automatic architecture reconstruction via static analysis, dynamic runtime analysis, or a mixture of both. A static example for SOA is the intelligent search support by Reichherzer et al. [65]. Their SOAMiner tool parses and analyzes common SOA artifacts like WSDL or BPEL files and conceptualizes knowledge relevant for architecture and maintenance from them. Similarly, Buchgeher et al. [15] provide a platform to provide up-to-date architectural information of large-scale service-based systems via extraction techniques using source code and other development artifacts like POM files. With five publications, this category was also especially popular for microservices. One example is the MicroART approach of Granchelli et al. [32] that combines static information from a code repository (e.g. Docker files) with dynamic runtime data collected via `tcpdump`. A second mixed approach is the MICROLYZE framework from Kleehaus et al. [40]. By combining data from service registries and OpenTracing monitoring with static build-time information, the system’s architecture can be continuously reconstructed while also taking infrastructure and hardware into account.

Overall, approaches in this category were concerned with providing accurate architecture documentation to increase analyzability and to ease maintenance efforts. Automation is used to reduce manual efforts, but this is challenging in heterogeneous and decentralized environments that consist of a very large number of diverse (micro)services. Most modern approaches combine static with dynamic analysis and sometimes even rely on tool-supported manual steps to increase accuracy.

**Model-Driven Approaches.** All 12 publications in this category were related to SOA and most of them were concerned with model-based verification during evolution. To support model evolution and change propagation across different model types such as business process or service models, Sindhgatta and Sengupta [71] proposed a framework that automatically analyzes Meta-Object Facility (MOF) compliant models and supports the selective application of changes to downstream models. Similarly, Liu et al. [48] designed a verification approach that is based on colored reflective Petri nets and simulates adaptive service evolution. A last approach in this area was created by Zhou et al. [88]: they analyze model consistency using hierarchical timed automata and also support the modelling of time constraints as well as architectural decomposition. In the area of business process management, Boukhebouze et al. [13] relied on rule-based specifications with the event-condition-action model to assess a business process’s flexibility and to estimate its cost of change. Rules are translated into a graph and subsequently used for analysis. Lastly, Lambers et al. [44] proposed a very early holistic approach for model-driven development of service-based applications. Their goal was to enable expert users to iteratively and rapidly develop flexible services through a series of models, code generators, and a graphical user interface for visual model creation and modification.

All in all, the presented approaches rely on different kinds of formal methods to enable faster and less error-prone development and evolution of service-based systems. Models specific to service orientation like business process models were frequently used. The predominant themes were consistency checking and verification.
Patterns. We identified 14 studies that discussed service-based design patterns, i.e. best practice solution blueprints for recurring design problems, in the area of maintenance and evolution. However, contrary to previous categories, most of them did not follow one major research direction. A few publications proposed new service-based design patterns beneficial for maintainability like the *Service Decoupler* from Athanasopoulos [4]. Others like Tragatschnig et al. [76] used patterns to describe and plan the evolution of service-based systems ("change patterns"), which should increase the efficiency and correctness of modifications. Some approaches centered around existing patterns in a system. Zdun et al. [85] provided a set of constraints and metrics for automatically assessing the pattern conformance in microservice-based system to avoid architectural drift. Demange et al. [19] designed the "Service Oriented Detection Of Patterns" (SODOP) approach to automatically detect existing service-based patterns via metric rule cards so that their design quality can be evaluated. Lastly, Palma et al. [55] analyzed the change-proneness of selected service-based patterns by studying an open source system. Using the metrics *number of changes* and *code churn*, they discovered that services with patterns needed less maintenance effort, but not with a statistically significant difference.

In general, this category was surprisingly heterogeneous with diverse pattern use cases, ranging from systematic maintainability construction to architecture conformance checking or service evolution description. Unfortunately, our SLR did not identify more holistic publications with collections and discussion of service-based patterns beneficial for maintainability, e.g. a literature study. Likewise, we only identified a single empirical study [55] that analyzed the impact of service-based design patterns on modifiability.

Antipatterns and Bad Smells. With 21 publications, this category was 50% larger than patterns, which could indicate that preventing suboptimal designs has been perceived as more important or as more related to maintainability than to other quality attributes. Research directions in this category were also not as scattered, but followed two main themes. The smaller group was concerned with the conceptualization of service-based antipatterns. Palma and Mohay [57] presented a classification taxonomy for 20 web service and Service Component Architecture (SCA) antipatterns and also defined metrics to specify their existence. In the area of microservices, Taibi and Lenarduzzi [75] synthesized 11 common bad smells by interviewing 72 developers. Similarly, Carrasco et al. [17] collected nine microservices architecture and migration smells by analyzing 58 sources from scientific and grey literature. The larger group in this category, however, went one step further and proposed automatic detection approaches for antipatterns. An example was the SODA (Service Oriented Detection for Antipatterns) approach from Nayrolles et al. [52], which the same authors later improved by mining execution traces [51]. For RESTful services, Palma et al. [56] proposed an approach which uses semantic as well as syntactic analysis to detect linguistic antipatterns. Lastly, Sabir et al. [69] combined both directions in their SLR that not only collected existing antipatterns, but also analyzed detection approaches.

The automatic detection of service-based antipatterns to efficiently identify design flaws was the prevalent theme in this category. Different approaches like static or dynamic analysis, machine learning, genetic programming, or combinations have been proposed. An understudied area, however, seems to be the systematic refactoring of detected antipatterns.

Service Identification and Decomposition. We identified 21 publications that focused on the activities of service identification or decomposition to increase maintainability. The majority of these (15) proposed a *process or method* to derive service candidates or to
decompose existing services or interfaces in a systematic or automatic fashion. A fully automated identification approach has been proposed by Leopold et al. [45]: they relied on semantic technologies to create a ranked list of service candidates from existing business process models. Athanasopoulos et al. [5] designed a tool-supported approach to analyze WSDL specifications with cohesion metrics. Based on the results, the existing interface is progressively decomposed into more cohesive units. Similarly, Daagi et al. [18] used a framework for Formal Concept Analysis (FCA) to identify hidden relations between WSDL operations and decompose the interface into several more fine-grained ones. Because numerous approaches have been proposed, there is also a decent amount of literature studies in this category. Kohlborn et al. [41] conducted a structured evaluation of 30 service analysis approaches and proposed a new consolidated method to address collected shortcomings. A second review by Cai et al. [16] tried to keep the literature analysis closer to the general software and service engineering process and derived common “high-value activities”. Lastly, a literature review from Bani-Ismail and Baghdadi [6] identified eight common service identification challenges. The same authors [7] also conducted another review to gather proposed evaluation frameworks for service identification methods.

Since a plethora of manual or automatic approaches has been proposed in this category, it becomes difficult to differentiate between them. Some publications tried to address this with literature studies, but selecting a feasible approach for a use case may still be challenging.

Maintainability Metrics and Prediction. Our second largest category consisted of 55 publications. Since existing source code or object-oriented metrics are only of limited relevance for service-oriented systems, most publications in this category proposed maintainability metrics specifically designed for service orientation. Some researchers approached this by focusing on a single maintainability-related design property like coupling [62], complexity [63], cohesion [61], or granularity [1]. Others tried to assemble holistic metric suites, like the SOA design quality model from Shim et al. [70] or the metrics suite from Sindhgatta et al. [72]. Because most proposed metrics were of an architectural nature and therefore difficult to collect from source code, some publications also focused on metrics for specific service-based artifacts like SoaML [31] or BPMN [74] diagrams. To create an overview and to compare proposed metrics, other researchers conducted literature studies. Nik Daud and Wan Kadir [54] collected and categorized service-based metrics according to structural attributes, applied phase, or artifact. Bogner et al. [11] targeted only automatically collectable metrics and also analyzed the applicability of SOA metrics for microservices. A few publications also used metrics and various machine learning techniques to predict the future maintainability of services. Wang et al. [81] used artificial neural networks to build prediction models of several web service interface metrics. In a slightly different fashion, Kumar et al. [43] applied feature selection techniques and support vector machines to evaluate the prediction quality of object-oriented metrics for the maintainability of service-based systems.

Overall, the main theme in this category was the definition of new or adapted service-based maintainability metrics. While many metrics have been proposed, their relevance and effectiveness often remained unclear. We identified only very few evaluation studies like the one from Perepeltchikov and Ryan [60]. Comparative literature studies were also rare. Furthermore, even though automatic collection was a prevalent topic and several tools like Q-ImPrESS from Koziol et al. [42] or MAAT from Engel et al. [22] were presented, the number of publicly available tools was very low, which could hinder replication studies and industry adoption. Lastly, there seems to be much potential in crossing boundaries between e.g. SOA, microservices, WSDL, or SoaML to adapt evaluated metrics to other fields.
Change Impact and Scenarios. A focus on dependencies between clients and services has made change impact analysis a popular service-based topic. This category also comprises publications about more general scenario-based maintainability evaluation. In total, we identified 39 papers, which made this the third largest category. A large number of these publications propose specific approaches. Wang and Capretz [82] combined information entropy with dependency analysis to quantify the relative importance of a service for change effects. Another approach is taken by Hirzalla et al. [34]: they created a framework (IntelliTrace), which relies on the modeling of traceability links between SOA artifacts like business goals, processes, or services. Based on these links, the impact of changes at different levels can be analyzed. Lastly, Khanh Dam [38] designed an approach based on association rule data mining to predict change impact using the version history of web services. To collect and compare proposed approaches, Amjad Alam et al. [2] conducted an SLR about impact analysis and change propagation for business processes and SOA. Their analysis of 43 studies concluded that very few mature approaches and tools existed, especially for bottom-up or cross-organizational analysis. The second major research direction was concerned with analysis of existing systems or APIs to derive information about their evolution change impact. Using a tool called WSDLDiff, Romano and Pinzger [67] extracted and analyzed fine-grained changes from the WSDL version histories of four web services from Amazon and FedEx. Similarly, Espinha et al. [24] analyzed the evolution of the Twitter, Google Maps, Facebook, and Netflix APIs. By interviewing six client developers and by analyzing source code version histories, they investigated how the API evolution affected service consumers. The two major research directions in this category were a) proposing approaches for change impact analysis and b) empirical studies on the evolution impact of existing service-based systems. Proposed methods were mostly based on dependency graphs or repository data mining. Artifacts specific for service orientation like WSDL files were often used. Very few publications, however, were concerned with general scenario-based evaluation, like the one from Leotta et al. [46], who used the Software Architecture Analysis Method (SAAM) to compare the maintainability of one SOA and one non-SOA alternative. Our review did not identify a scenario-based method specifically designed for service orientation.

Evolution Management. With 56 papers, evolution management was our largest category (25%). Since it was also our most general one, it consisted of diverse approaches to control and plan service evolution. Therefore, no major research directions could be identified. However, one similarity among these publications was that most of them (48 of 56) proposed either a process or method or a model or taxonomy, i.e. most work was conceptual in nature. To illustrate this diversity, we present some selected approaches. Zhang et al. [86] designed a framework to manage requirements evolution in service-based system. The framework is based on Role, Goal, Process and Service (RGPS) elements and also contains a meta model and strategies. Another model and methodology was proposed by Zuo et al. [90]. In their change-centric model for web service evolution, they specify e.g. stakeholder behavior, service versioning, and the details of service changes. Feng et al. [25] created a taxonomy framework for SOA evolution that describes the motivation, location, time, and support mechanism of changes. The goal of their work is to support the analysis and planning of service evolution. Lastly, a more holistic framework for web service evolution support (WSDarwin) was presented by Fokaefs [27]. WSDarwin consists of an Eclipse plugin for automatic service client adaptation on interface changes, a web application to automatically generate WADL documentation for RESTful services and to compare different WADL versions, and a decision support system for evolving service ecosystems based on game theory (see also [28]).
4.4 Differences Between Approaches for SOA and Microservices (RQ4)

When analyzing differences between the identified publications for SOA and those for microservices, the most apparent finding was the small percentage of microservice-focused studies (less than 11% or less than 25% for 2014-2018). While microservices are the younger paradigm, it seems that the scientific interest in their maintainability assurance is just getting started. Possible reasons could be that most microservice-based systems are still fairly young and therefore decently maintainable or that their inherent evolution qualities are perceived as more beneficial when compared to SOA. With respect to thematic categories (see Fig. 11), we see three categories without approaches exclusively designed for microservices (evolution management, change impact & scenarios, and model-driven approaches), but only one without a single publication on microservices, namely model-driven approaches. While such approaches do exist for microservices [64], they were not identified by our review, maybe because they are not advertised for maintainability. The most prominent category for microservices was architecture recovery & documentation (5 of 9 papers, all of them for recovery), which highlights the importance of this topic. While automatic microservice decomposition and extraction is also a popular topic in academia [30], only two of the 21 papers in service identification & decomposition were about microservices. This is mainly due to the fact that we excluded pure legacy migration approaches. Lastly, for antipatterns & bad smells, publications for microservices were mostly concerned with defining antipatterns while more established SOA publications already proposed automatic detection approaches. This may be a sensible next step for microservices.

![Figure 11 Distribution: Thematic Categories Grouped By Architectural Category.](image)

In general, we identified a lot of potential for the adoption of SOA approaches for microservices, especially in the areas of maintainability metrics & prediction, antipatterns & bad smells, service identification & decomposition, and patterns. Existing SOA research in these categories could be valuable for the evolution and maintenance of microservices if the techniques are also applicable for e.g. strong decentralization or technological heterogeneity. Moreover, the majority of studies on RESTful services should be directly applicable for microservice-based systems. Lastly, our SLR did not identify studies about possible negative maintainability impacts of microservices, e.g. in the areas of knowledge exchange, team synchronization, technological heterogeneity, or code duplication. While the maintainability of microservices as an architectural style is generally perceived as positive, we still see the potential for empirical studies on these topics.
5 Threats to Validity

Results derived from systematic literature studies may suffer from limitations in different areas if not performed with great rigor (see e.g. [87]). Even though we adhered to a detailed SLR protocol and the complete study selection and categorization was performed by two researchers to mitigate subjective bias, there is still the possibility for threats to validity. One example for the planning phase is that our search strategy could have been insufficient due to missing keywords or not included databases. Likewise, the presentation of exemplary publications to highlight existing directions per category was subject to our own perception of relevance. Other researchers may disagree or come to different conclusions. However, the most prominent threat to validity with this SLR is most likely the limiting of search results to the first 250 entries per publisher (1000 papers from four publishers). This made the results dependent on the relevance sorting of each search engine and may also hinder replicability if publishers decide to change their algorithms in the future. Even though our snowballing nearly doubled the amount of selected primary studies, there is still the possibility that we may have missed relevant literature branches without links to our initial set. We accepted this threat to keep the effort manageable within the project time frame.

6 Conclusion

Since the scientific literature on maintainability assurance for service-oriented systems is diverse and scattered, we conducted an SLR with the goal to categorize the proposed approaches and to analyze differences between SOA and microservices. As an answer to RQ1, we derived a categorization set with architectural (SOA, microservices, both), methodical (method or contribution), and thematic (subfield of maintainability assurance) categories from the 223 selected primary studies. The distribution analysis (RQ2) revealed for example that nearly 90% of papers exclusively targeted SOA (199) and that evolution management and maintainability metrics & prediction were the most prominent thematic categories (both with ∼25%). For each thematic category, we also presented the most relevant research directions with illustrating studies (RQ3). Exemplary differences between approaches for SOA and microservices (RQ4) were the importance of architecture reconstruction and the absence of automatic antipattern detection approaches for microservices. While there was only a small number of approaches for microservices, we identified a lot of potential for adapting SOA approaches in several categories.

Future research could be concerned with literature studies for individual categories to provide more insights into these subfields and to analyze the adoption potential for microservices in greater detail. An analysis of maintainability-related differences between orchestration (SOA) and choreography (microservices) across the primary studies may also yield helpful results for the usage of these two paradigms. Lastly, it could be interesting to replicate this SLR exclusively for microservices in a few years when more publications exist for these topics. To enable replication and to allow convenient reuse of our results, we shared all SLR artifacts in a GitHub repository2.

2 https://github.com/xJREB/slr-maintainability-assurance
SLR on the Maintainability Assurance of Service- and Microservice-Based Systems


58 Francis Palma, Mathieu Nayrolles, Naouel Moha, Yann-Gaël Guéhéneuc, Benoit Baudry, and Jean-Marc Jézéquel. SOA antipatterns: an approach for their specification and detection.


