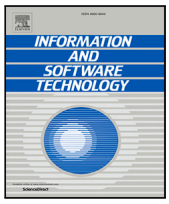
Information and Software Technology 188 (2025) 107870

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Microservices testing: A systematic literature review

Francisco Ponce [a,*], Roberto Verdecchia [b], Breno Miranda [c], Jacopo Soldani [a]

[a] University of Pisa, Pisa, Italy
[b] University of Florence, Florence, Italy
[c] Federal University of Pernambuco, Recife, Brazil

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.16941552

Keywords:
Microservices
Microservice architecture
Software testing
Functional testing
Non-functional testing
Testing strategies
Systematic literature review

ABSTRACT

Context: Microservices offer scalability and resilience for modern cloud-native applications but present significant challenges in software testing due to their distributed and heterogeneous nature.
Objective: This study aims to consolidate and classify the current body of knowledge on microservice testing through a systematic literature review, providing actionable insights for both researchers and practitioners.
Methods: Following established guidelines for systematic literature reviews in software engineering, we identified 74 primary studies relevant to microservices testing. These studies were systematically categorized using the SWEBOK (*Software Engineering Body of Knowledge*) taxonomy for software testing. Specifically, we classified the identified techniques according to their testing objectives, levels, strategies, and types. We also evaluated the study types to gauge the maturity and readiness of the current state-of-the-art in microservice testing.
Results: System testing emerged as the most frequently investigated testing level, followed by integration, unit, and acceptance testing. Conformance, regression, and API testing were the most common functional testing objectives, while performance efficiency and reliability were instead predominant in the case of non-functional testing. Specification-based testing strategies were the most adopted, followed by usage-based and fault-based ones. Additionally, most studies employed laboratory experiments and had low-to-medium technology readiness levels, indicating early-stage maturity. The systems under test varied in size and domain, with TrainTicket being the most widely used reference benchmark for large systems.
Conclusion: While significant progress has been made in microservice testing, the field remains fragmented, with notable gaps in areas such as, e.g., flexibility and security testing. The dominance of early-stage proposals highlights the need for more empirical validation and industry-grade benchmarks to facilitate broader adoption. This review offers a structured roadmap for future research and practical adoption in microservices testing.

1. Introduction

Microservices (MSs) enable realizing cloud-native applications [1], bringing various advantages such as fault resilience and scalability [2]. This motivates why many IT companies (e.g., Amazon, Meta, Netflix, and Spotify) already deliver their core business through microservice-based architectures (MSAs) [3].

The advantages of MSAs come at the price of some *pains*, one of the most prominent being the inherent complexity of testing MSAs [4,5]. An MSA is composed of various heterogeneous services that interact to deliver the application's business. The number of services forming an MSA, together with their interactions, gives rise to complex software architectures [6]. This in turn makes it challenging to test MSAs, especially at the level of system and integration tests. Concrete examples of difficult testing tasks are, e.g., measuring the performances of interacting services or assessing the overall user experience from the frontend of an MSA [5].

Existing solutions for testing MSAs are, however, scattered across different pieces of literature. They also often focus only on specific testing levels (i.e., unit, integration, or system testing), as well as on specific testing objectives, e.g., conformance, regression, or performance testing. This fragmentation of the body of knowledge on MS testing poses challenges for practitioners seeking to implement a robust testing strategy for their MSAs, as well as for researchers wishing to contribute and advance the state-of-the-art in MS testing.

To this end, this article aims at systematically reviewing the existing solutions for testing MSAs. By relying on established protocols for

* Corresponding author.
E-mail addresses: francisco.ponce@di.unipi.it (F. Ponce), roberto.verdecchia@unifi.it (R. Verdecchia), bafm@cin.ufpe.br (B. Miranda), jacopo.soldani@unipi.it (J. Soldani).

https://doi.org/10.1016/j.infsof.2025.107870
Received 7 May 2025; Received in revised form 17 July 2025; Accepted 7 August 2025
Available online 23 August 2025

conducting systematic literature reviews (SLRs) in software engineering [7,8], we systematically identified 74 primary studies contributing to the realm of MS testing, which we then classified by relying on the SWEBOK (*Software Engineering Body of Knowledge* [9]) taxonomy for software testing. More precisely, we classified the existing MS testing techniques based on the testing objective, level, strategy, and type, while also assessing the study types to assess the readiness of the state-of-the-art in MS testing. This article reports on the results of our systematic review by also discussing the main findings on the 4W1H (i.e., *what*, *when*, *where*, *why*, and *how*) of testing MSAs.

We believe that the SLR presented in this article can provide benefits to both practitioners and researchers working with MSAs. We indeed not only help them in finding the testing techniques most suited to the needs of their MSAs, but we also discuss some open challenges and possible research directions on MS testing.

The rest of this article is organized as follows. Sections 2 and 3 provide the necessary background and discuss related work, respectively. Section 4 illustrates the SLR process we followed, aiming at encouraging its repeatability. Sections 5 and 6 report on the identified techniques for functional and non-functional testing of MSAs, respectively. Section 7 discussing the 4W1H of testing MSAs. Finally, Sections 8 and 9 discuss threats to validity and draw some concluding remarks, respectively.

2. Background

2.1. Microservices

Microservices were first introduced in 2014 by Lewis and Fowler in their influential blog post [10], where they outlined an architectural style for developing applications as collections of small, independent services. Each microservice is designed around a specific business capability, operates in its own process, and communicates with other services within the application via lightweight protocols (such as HTTP APIs). These services are typically organized around business functions and can be developed, deployed, and scaled independently, in contrast to monolithic systems where the entire application is tightly integrated [11]. This architectural approach can be viewed as an evolution of Service-Oriented Architecture (SOA), which emphasizes service independence, self-management, and lightweight interaction [12]. Microservices are inherently loosely coupled, enabling independent deployment through automated, often containerized, platforms [13]. This decentralization supports key characteristics such as continuous delivery and the use of lightweight protocols for inter-service communication. This independence in deployment supports scalability and flexibility, making microservices a natural fit for service-oriented architectures [14].

2.2. Software testing

Software testing is a crucial activity within the software development lifecycle, aimed at verifying that a system functions correctly under various conditions. In this work, we categorize the primary studies using the SWEBOK taxonomy for software testing. Next, we provide an overview of the key test levels, techniques, and objectives, as outlined in the SWEBOK [9].

2.2.1. Test levels

The SWEBOK defines four test levels, each targeting different scopes and objectives [9]. *Unit Testing* focuses on verifying the correctness of individual components or subprograms in isolation. It is often conducted by developers to ensure that each unit performs as intended. *Integration Testing* examines the interaction between integrated units or components. It aims to uncover interface errors and assess interoperability between modules or systems. *System Testing* validates the complete and integrated system against specified requirements. It addresses both functional and non-functional aspects, such as performance, reliability, and security. *Acceptance Testing* determines whether the software meets user needs and requirements. This testing level is typically carried out by end users or clients, focusing on usability and operational validation to ensure the software is fully ready for deployment.

2.2.2. Test techniques

A broad spectrum of test techniques supports the selection and design of effective test cases [9]. *Specification-based* techniques (black-box testing) derive test cases from requirements and functional specifications (e.g., equivalence partitioning, boundary value analysis). *Structure-based* techniques (white-box testing) leverage knowledge of the software's internal structure (e.g., statement and branch coverage). *Experience-based* techniques rely on testers' intuition and prior experience (e.g., error guessing, exploratory testing). *Fault-based and mutation techniques* simulate or inject faults to assess the sensitivity and fault-detection power of test suites. *Usage-based* techniques model real-world usage to evaluate reliability and performance under operational conditions.

2.2.3. Objectives of testing

Testing can serve multiple objectives depending on context and development phase. *Conformance Testing* verifies that a software system adheres to predefined specifications, standards, design rules, or coding guidelines. It ensures that the implementation correctly follows formal definitions such as communication protocols, file formats, or industry standards. *Regression testing* focuses on confirming that recent changes to the codebase have not unintentionally affected existing functionality. It involves re-executing previously passed test cases to detect any newly introduced defects. *Interface and Application Program Interface (API) Testing* aims to validate the communication between different software components or systems, ensuring correct data exchange and interaction behavior. For APIs, this involves testing endpoint responses, parameter handling, error conditions, and compliance with interface specifications. This type of testing is crucial in modular and service-oriented architectures, where robust and predictable interfaces support system integration, scalability, and external interoperability.

For a more comprehensive overview of additional objectives of testing, we refer the reader to the SWEBOK [9].

2.3. ISO/IEC 25010 software quality standard

Software quality models are frameworks that define, measure, and assess the quality of software products. These models identify key attributes of software quality and establish criteria for evaluating whether a software system meets specific standards. To classify primary studies focused on non-functional testing in terms of testing objectives, we rely on the ISO/IEC 25010 standard [15]. This standard structures software quality into nine high-level attributes: functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety.

Performance efficiency, reliability, and flexibility are the quality attributes discussed in the primary studies selected for this work. *Performance Efficiency* relates to the system's resource usage and responsiveness under specified conditions. Key aspects include time behavior, resource utilization, and capacity. *Reliability* measures the system's ability to perform consistently over time. This includes fault tolerance, availability, and recoverability. *Flexibility* reflects the adaptability of the system to different environments or requirements, including adaptability, scalability, installability, and replaceability.

A comprehensive introduction to the other existing quality attributes can be found in the ISO/IEC 25010 standard [15].

3. Related work

Our SLR aims at complementing the few existing secondary studies on MS testing [16–20]. These prior works cover different angles of MS testing, but with different scope, depth, and/or recency of the considered primary studies, and this mainly motivates our SLR.

Among the most recent works is the systematic mapping study (SMS) by Hui et al. [18], which analyzed literature published up to early 2024. Their SMS provides a broad overview by mapping primary studies to the testing methodologies they employ. While valuable, it focuses primarily on high-level classification of testing approaches and does not delve into detailed testing attributes such as testing levels, objectives, or strategies. Moreover, Hui et al. [18] do not assess the technological maturity of proposed techniques, nor do they analyze systems under test (SUTs). Our SLR complements the SMS by Hui et al. [18] by going into the details on the 4W1H of MS testing, which comprise testing levels, objectives, and strategies, as well as the technological maturity of proposed techniques and SUTs.

Similar arguments apply to the SMS by Waseem et al. [20], who classify 33 primary studies published between 2015 and 2019. Waseem et al. [20] also provide a high-level classification of testing approaches, focusing on less recent studies, and still without delving into detailed testing attributes, e.g., testing levels, objectives, strategies, or SUT. Our SLR complements the SMS by Waseem et al. [20] by providing a more up-to-date coverage of the state-of-the-art on MS testing, and by going more in-depth with testing attributes. We indeed analyze testing levels, objectives, and strategies, as well as the technological maturity of proposed techniques and SUTs.

Ghani et al. [17] performed an earlier SLR limited to studies published between 2010 and 2018, covering only 15 primary studies. Although their focus on testing objectives (specifically, quality attributes) is valuable, the restricted temporal scope and dataset size significantly limit its relevance today, especially given the exponential growth of research in this field since 2018. We complement the SLR by Ghani et al. [17] by providing a more up-to-date analysis of the state-of-the-art on MS testing, as well as by going beyond testing objectives, e.g., by also covering testing levels, strategies, and SUTs.

Finally, Fischer et al. [16] and Simosa and Siqueira [19] focus only on one of the testing attributes out of those covered by our SLR. Particularly, Fischer et al. [16] provide an SMS on SUTs, identifying 134 systems that can be used for MS testing and monitoring. Simosa and Siqueira [19] instead surveyed contract testing in microservices between 2015 and 2022, emphasizing techniques and tools related to contract-based validation. The specific focus of both Fischer et al. [16] and Simosa and Siqueira [19], however, makes their studies complementary rather than redundant with ours.

In summary, compared to the existing secondary studies on MS testing [16–20], our SLR offers a more detailed, comprehensive and/or up-to-date perspective by analyzing 74 primary studies published up to the end of 2024. We uniquely classify testing approaches using the SWEBOK taxonomy, assess their technological maturity, and identify reference SUTs. Additionally, our SLR identifies up-to-date research challenges. These contributions collectively complement the results available in existing studies and establish the distinct value of our work in comparison to prior studies.

4. Research design

Using the Goal-Question-Metric approach [21], the research objective of our investigation can be formulated as follows:

Analyze *testing approaches*
For the purpose *of knowledge collection and categorization*
With respect to *academic literature*
From the viewpoint of *researchers*
In the context of *microservice-based systems.*

In order to collect the data for our literature review, we follow the systematic process outlined by Kitchenham [7]. In order to systematically identify a set of primary studies, we follow the approach presented by Wohlin [8], which entails a bidirectional snowballing search based on a starting set obtained via an automated search query executed on Google Scholar. The use of Google Scholar allowed us to avoid bias in favor of any specific publisher [8]. Further threats related to the use of Google Scholar are discussed in Section 8.2.

A high level overview of our research process is depicted in Fig. 1 and is further detailed in the following sections.

4.1. Phase 1a: Automated literature search

As initial step for our primary studies selection, we collect an initial set of potentially relevant studies *via* the execution of an automated query on the Google Scholar literature indexer. The title-focused automated search was designed based on our research objective, and with the aim to gather related literature that explicitly focuses specifically on the topic considered while being as encompassing as possible. Based on this rationale, the query utilized results in the exact following search string that is utilized to identify the initial set of potentially primary studies:

```
ALLINTITLE: ("microservice*" AND "test*")
```

Executing the automated search results in the identification of 186 potentially primary studies, which are then manually filtered *via* a pre-defined set of selection criteria in a later phase (see Phase 4 Section 4.4).

The query was executed on the 16th of December 2024, and to be as inclusive as possible left unbounded the publication date range. The results were sorted alphabetically to mitigate possible biases introduced by potentially confounding search ranking algorithms. To be as inclusive as possible and mitigate potential threats to internal validity, no cutoff point was utilized, i.e., the entirety of the query results was considered for manual selection (see also following phases).

4.2. Phase 1b: Supplementary automated query

In order to mitigate potential threats to internal validity related to the use of a single literature indexer, we execute an additional automated query by leveraging the Scopus digital library.[1] A query identical to the one utilized in Phase 1 is adopted to automatically search potentially primary studies on our second source of data, namely the Scopus digital library.

As for the primary query, the supplementary one was executed on the 16th of December 2024, and to be as inclusive as possible left unbounded the publication date range. The query execution resulted in the identification of 278 potentially primary studies.

4.3. Phase 2: Duplicate removal

Merging the results of the automated literature search queries executed in Phase 1a and Phase 1b, we remove the duplicates of the two queries by relying on the potentially primary studies metadata, namely publication title and publication venue. This process results in the removal of 92 primary studies. More specifically, the potentially primary studies identified *via* the Scopus platform result to be a subset of the items identified *via* the Google Scholar literature indexer. As a final output of this research step, we identified 186 potentially primary studies.

[1] https://www.scopus.com/. Accessed 3rd April 2025.

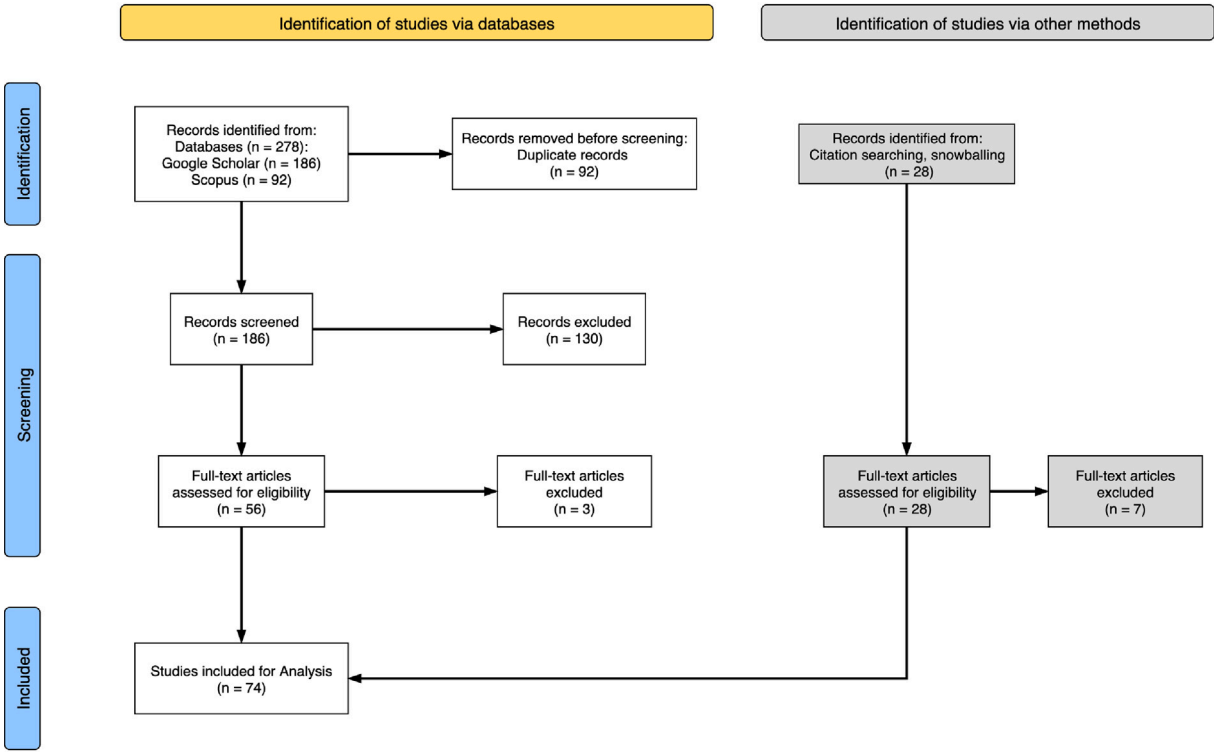


Fig. 1. Systematic literature review process illustrated through a PRISMA flow diagram [22].

4.4. Phase 3: Manual selection

Following the consolidation of the potentially primary studies set identified automatically, we conduct a manual selection process of the primary studies based on a pre-defined list of inclusion and exclusion criteria. A potentially primary study is selected in our SLR if it satisfies all of our inclusion criteria (I) and none of the exclusion ones (E). The selection criteria we use are as follows:

- I1: The paper focuses on microservices.
- I2: The paper focuses on software testing.
- I3: The paper proposes a solution for testing microservices.
- E1: Non-English publications.
- E2: Publications for which the full text is not available to us.
- E3: Duplicates of already included publications.
- E4: Secondary or tertiary studies.
- E5: Publications in the form of editorials, tutorials, books, extended abstracts, etc.
- E6: Non-scientific publications or grey literature.

The first two inclusion criteria (I1, I2) are designed to identify primary studies that focus on the investigated topic, namely microservice, and hence present relevant data for our literature review. The third inclusion criteria instead (I3) is utilized to select exclusively studies that present an approach to test microservices. This latter inclusion criteria allows our investigation to gain a comprehensive overview of the existing solutions to test microservices, by excluding adjacent topics related to microservice testing, e.g., experience reports. The exclusion criteria instead are utilized to ensure that we are able to extract data from the papers (E1, E2), we do not include redundant information (E3, E4), and consist of scientific literature (E5, E6). As further clarifications on some selection criteria, E2 considered manuscript that are indexed but not available online. Regarding E3 instead, in case the extension of an already included paper is identified, we included only the extended study (instead of the original one) as, by nature, extensions contain both the original content and some additional ones.

The initial set of 186 studies is inspected by the four authors, with each author analyzing for inclusion a distinct subset of potentially primary studies. To ensure agreement among reviewers in the selection process, the inter-rater agreement is calculated *via* Fleiss' kappa on a subset of 25 studies. This process results in a Fleiss' kappa of 0.78, demonstrating a substantial agreement among reviewers. As an additional measure to prevent potential threats to internal validity, weekly meetings are held to jointly discuss doubts, corner cases, and impediments to further strengthen the alignment among reviewers.

The manual selection process terminates with the selection of 56 primary studies to be used for the follow-up snowballing process and full-text assessment (see Fig. 1).

4.5. Phase 4: Snowballing

To complement our preliminary set of primary studies identified *via* the automated literature search, in line with common systematic literature review practices Wohlin [8], we adopted a forward and backward snowballing process. During this phase, both the paper cited by the already included primary studies and the ones citing them are inspected for inclusion in our literature review. The snowballing process proceeds iteratively in multiple rounds, i.e., the primary studies included *via* snowballing undergo a follow-up snowballing process to include new literature. As a stopping criterion, we adopt theoretical saturation, i.e., our snowballing process terminates when in a new round of snowballing no new papers are selected for inclusion.

The four authors conduct the snowballing simultaneously on a distinct subset of the primary studies. While the rater alignment is already established *via* inter-rater agreement calculation of the previous phase, as a mitigation strategy to potential internal threats to validity, weekly meetings are held to discuss doubts and further align the selection process among reviewers.

The snowballing process terminates with the inclusion of 21 new primary studies, leading to a total number of primary studies considered in this systematic literature review equal to 74.

Table 1
Classification of the available functional testing techniques based on testing objective and testing strategy.

	Specification-based	Structure-based	Usage-based	Experience-based	Fault-based
Regression	[25–34]	[33,35,36]	[36,37]		[38]
Conformance	[39–52]	[53–55]	[55,56]	[57]	[58]
API	[59–61]	[62]			

4.6. Phase 5: Data extraction and synthesis

To extract the data from the identified primary studies, we adopt a mix of deductive coding (DC) combined with open coding (OC) followed up when necessary by an axial coding process (AC) [23]. In the following, we report the data attributes collected from each primary study, associated with the coding strategy adopted to extract it.

To understand the publication trends of the microservice testing literature we consider as demographic data the *Publication year* (DC), and publication venue, namely *Journal*, *Conference* or *Workshop* (DC).

At a more semantic level, we extract from the primary studies (i) the study type by following the characterization of Stol and Fitzgerald [24] (DC), the testing level as defined in software testing of the SWEBOK [9] (DC), if the testing approach regards one or more functional or non-functional requirements (DC), in case the non-functional requirement(s) considered for testing, as documented in the ISO/IEC 25010 [15] (DC), and the testing type, that could either assume white-box, gray-box, or black-box values (DC).

To gain a deeper understanding of the microservice testing approaches reported in the primary studies we also extract from the primary studies the testing objective as defined in the SWEBOK [9] and complemented *via* additional codes when necessary (DC, OC, and AC), the testing strategy as categorized in the SWEBOK [9] (DC), and the method used to validate the testing approaches, that could either be in-vitro, in-vivo, in-silico, or in-virtuo (DC). As an additional attribute, we also consider if an implementation or tool of the testing approach is made available or not (DC).

Finally, we complement our data extraction process by considering also the nature of the software under test (SUT) used to evaluate the approaches. To this end, we extract from the primary studies the number of SUT considered (DC), the SUT size in terms of the number of microservices (DC), and the domain of the SUT, e.g., transportation or e-commerce (OP and AC).

The data extraction process leads to the scrutiny of the 74 primary studies considered in this research according to the 15 aforementioned attributes, leading to a total dataset of 1.1k distinct values. The values are then analyzed *via* simple summary visualizations (e.g., barplots and heatmaps) and statistical analyses (e.g., percentages) to present the collected data in a comprehensive, clear, and objective fashion.[2]

5. Functional testing

Functional testing ensures that MSAs meet their behavioral requirements, e.g., verifying their functional correctness and/or compliance with specifications. The functional testing techniques proposed in the selected studies are classified in Table 1, providing a structured overview of their focus areas. The following sections present these techniques, organized according to the specific functional objectives they target.

[2] A replication package for our study is available at https://doi.org/10.5281/zenodo.16941552.

5.1. Regression testing

Regression testing result to be a frequent objective covered by MS testing literature [25–38]. Among testing strategies adopted in MS regression testing research, the majority focuses on specification-based ones [25–34]. As example, the work of Elsner et al. [28] present `microRTS`, an adaptation of dynamic regression test selection tuned to be applicable in an end-to-end MS testing scenario. `microRTS` is evaluated by considering a case study considering 12 different versions of an application. By filtering in a semi-automated fashion test traces, the approach resulted able to reduce testing effort up to 50%. Kargar and Hanifizade [31] instead consider at large the topic of automating regression testing in MSAs. Specifically, the Kargar et al. study how MS regression testing process could be integrated in continuous delivery environments by considering a specific set of technologies, such as GitLab, Jenkins, Docker, and Kubernetes. As other work using specification-based testing techniques, Chen et al. [26] present the test prioritization approach CIPC. CIPC used as underlying theoretical basis for test prioritization the concept of belief propagation, that is, a repetitive process for estimated interpretation based on graph structure. In CIPC, belief propagation determines test case prioritization by analyzing code coverage changes between different versions of the MS system. Another specification-based techniques leverages the graph-like nature of MSAs [25]. Specifically, in the work of Chen et al. [25], a PageRank algorithm is used to prioritize test cases based on single-objective and multi-objective strategies. The rank of the test cases to be executed is calculated based on API gateway logs (similar to the test selection approach of Chen et al. [35]). Experimentation on four MS-based systems showcase the proposed approach to be twofold more effective than random prioritization. In another work utilizing the graph-like structure of MS interdependencies, Ma et al. [32] propose an approach enabling to reconstruct MS dependencies and identify test associated to code changes. The MS dependencies reconstruction process leverages service invocation chains, which are constructed by utilizing the Java Reflection mechanism, enabling to obtain information pertaining microservice endpoints and service calls. At a higher level of abstraction, De Angelis et al. [27] address the challenge of discovering relations between test programs in MSAs. The approach of De Angelis et al. is based on symbolic execution of test programs, which is used to collect information about the invocations of local and remote APIs performed when running MS programs. The information collected is then processed *via* a rule-based automated reasoning engine, enabling to infer dependencies and similarities among different test suites. Schneider et al. [33] presents a systematic test concept for developing MSAs which covers different test types, from end-to-end to unit tests. The test concept considers the entire test pyramid as part of the microservice engineering process. In the proposed approach, the acceptance criteria is specified as Gherkin features according to BDD practices, which are used for the development of end-to-end tests. Considering a different perspective, Gazzola et al. [29] present `ExVivoMicroTest`, and approach enabling to generate test cases for future versions of microservices. `ExVivoMicroTest` is based on lightweight execution monitoring and tracing to reconstruct how services are used in production. By processing the obtained execution traces, tests are then generated and selected based on their relevance. Finally, two papers utilizing on regression testing by using a specification-based technique focus also on performance efficiency

of MSAs [30,34]. Specifically, Janes and Russo [30] present PPTAM+, a tool designed to continuously monitor the degradation of a system during its refactoring to a MSA. The tool, designed to be integrated in a DevOps pipeline, identifies MS experiencing performance degradation issues by comparing a reference operational profile against the runtime performance collected *via* regression performance tests. The work of Smith et al. [34] instead present a more meta-scientific contribution, by presenting a test benchmark to evaluate the effectiveness and efficiency of regression testing approaches. The benchmark includes full functional regression testing coverage for two open-source MSAs, and is designed to support researchers by providing the necessary tools to evaluate regression testing approaches in a systematic and repeatable manner.

Only two papers which consider regression testing are based on an usage-based testing strategy [36,37]. Cooper et al. [37] a usage-based test selection approach by focusing budget aware selection, i.e., test selection based on predefined budget constraint, such as limited time, resources, or computational costs. The budget aware selection is based on the collection of real-world usage traces, which are then utilized to identify and execute a subset of tests. De Angelis et al. instead use a similar strategy by combining however concrete and symbolic execution to support regression testing processes De Angelis et al. [36]. The resulting traces are then processed in a similar fashion of the work by De Angelis et al. [27] by leveraging rule-based automated reasoning. A set of similarity criteria can then be used to identify the subset of test cases to be run in a regression testing scenario.

Finally, two researches are the only ones which use different testing strategies from all the others, namely structure-based [35] and fault-based testing strategies [38]. Chen et al. [35] propose `MRTS-BP`, a test selection approach designed to be as lightweight as possible on development processes. Considering high integrability, `MRTS-BP` utilized as input exclusively API gateway logs, which are used to mine MS dependencies and change impacts based on a propagation calculation. As the work of Chen et al. [26], the approach is based on the concept of belief propagation, that is, a repetitive process for estimated interpretation based on graph structure. At the intersection of MS regression testing and software reliability instead, He et al. [38] present `ResilienceGuardian`, a fault-based framework designed to detect erroneous software changes that reduce fault resilience. The framework is based on a classification algorithm that is trained on a set of synthetically injected faults. The resulting model is then utilized to classify code changes that are more probable to introduce faults in an MSA.

5.2. Conformance testing

MSA conformance testing is mainly supported through *specification-based* strategies [39–52]. In this context, some studies have focused on event-driven microservice testing, which validates the behavior and interactions of microservices communicating via asynchronous events. For instance, Ma et al. [47] presented a unit testing strategy, considering message brokers and the saga design pattern. The proposed approach enables the creation of unit tests that comprehensively identify errors in publishing, channel subscriptions, message formatting, and error handling.

Specification-based strategies also facilitate fault diagnosis, the process of identifying, classifying, and localizing failures in MSAs. Notably, Zhang et al. [52] introduced SynthoDiag, a fault diagnosis framework for microservices that automates fault diagnosis using multi-source logs and a knowledge graph. Long et al. [44] presented IntelliFT, a fitness-guided resilience testing technique designed to uncover defects in fault-handling logic. Likewise, Heorhiadi et al. [42] introduced Gremlin, a failure injection framework that tests fault-handling capabilities by manipulating inter-service messages.

Furthermore, automatic testing has also been explored through *specification-based* strategies. For example, Li et al. [43] explored API automation testing for microservices, incorporating machine learning techniques to generate high-fidelity test data based on historical usage patterns, improving regression testing and overall application quality. Similarly, Ding et al. [40] proposed an automatic test data generation method for microservice applications, leveraging extended service interface descriptions and an improved pairwise algorithm to enhance testing efficiency. Rahman and Gao [48] introduce RAATA, a reusable automated testing architecture for MS within the context of Behavior-Driven Development (BDD). The proposed approach outlines how BDD artifacts should be organized to facilitate the reuse of acceptance tests across multiple repositories, thereby reducing the maintenance burden on developers and testers. Instead of spreading BDD features/scenarios across multiple microservices repositories, a new repository called Automated Acceptance Tests (AAT) repository is introduced. An AAT repository typically has top level directories including features, which contains subdirectories for each microservice, and step-implementation. Duan et al. [41] instead propose a decision tree algorithm to classify and filter test cases, enhancing automation and efficiency.

Finally, Ayas et al. [39] provide another example of *specification-based* MSA conformance testing, with a different objective than those described above. They analyzed 16 GitHub repositories to understand how these projects cover different testing levels. The contribution by Ayas et al. [39] drafts a testing architecture, outlining activities and artifacts, and demonstrates how these align with established best practices.

MSA conformance can also be tested via other strategies than those specification-based. For example, Wu et al. [55] combine *usage-based* and *structure-based* testing strategies to test the conformance of microservices. More precisely, Wu et al. [55] introduce CCTS (Composite Contract Testing Service), a tool that integrates consumer-driven contract testing with state models to validate event exchanges and detect potential defects like isolated states, cyclic dependencies, and unqualified event sequences. CCTS analyzes event logs to verify compliance with expected state transitions, enhancing the reliability of event-driven microservices.

The above-mentioned strategies are also used standalone to test MSA conformance. Particularly, Ma et al. [56] and Abdelfattah et al. [53] focus on coverage aspects in MSA conformance testing, respectively adopting *usage-based* and *structure-based* testing strategies. More precisely, Ma et al. [56] propose a tool for analyzing and visualizing service dependencies in MSAs, which also improves test coverage by identifying untested service calls and automatically generating test cases. Abdelfattah et al. [53] instead propose test coverage metrics for MSA conformance testing, leveraging both static and dynamic analysis to map tests to microservice endpoints.

Vassiliou-Gioles [54] provide another example of *structure-based* MSA conformance testing, however with a different objective. They propose instance identification as a method to enhance integration testing by addressing managed service ambiguities, e.g., unknown contributors or temporary blindness. Their ultimate goal is to enable tracking of service versions and execution instances, ensuring that test results accurately reflect the tested components. By enriching service communication with structured metadata, the approach proposed by Vassiliou-Gioles [54] aims at improving observability, namely allowing testers to verify that the expected service instances contribute to test execution.

Different testing strategies are instead enacted by Almutawa et al. [57] and Yao et al. [58], who use *experience-based* and *fault-based* testing, respectively. Particularly, Almutawa et al. [57] propose Kashef, an LLM-assisted tool, which employs communicative agents to iteratively generate and execute system tests, leveraging LLMs for reasoning and code generation. Yao et al. [58] presents a method for fault-based testing in MSAs, aiming to infer fault probabilities even under sparse tracing conditions. These probabilities are then used to estimate missing fault propagation details, enabling effective fault localization without requiring manual annotation or dense traces.

Table 2
Classification of the available non-functional testing techniques based on targeted quality attribute and testing strategy.

	Specification-based	Structure-based	Usage-based	Experience-based	Fault-based
Flexibility			[65]	[66]	
Performance efficiency	[30,34,67–79]	[80,81]	[37,82,83]	[66]	
Reliability	[84–86]		[82,87,88]		[38,89–95]

5.3. Interface and application program interface testing

API testing in MSA is conducted primarily using *specification-based* testing strategies, where test cases are derived from the software specifications or requirements [59–61]. For example, Rattanukul et al. [59] introduces Microusity, a tool designed to perform RESTful API testing for a specific microservice pattern known as backends for frontends (BFF). Microusity relies on the OpenAPI specification of the APIs to generate test inputs and identify errors based on HTTP response status codes. Ashikhmin et al. [60] propose a method to streamline the generation of mocks for REST services using RESTful API Modeling Language (RAML) specifications. Similarly, Lin et al. [61] proposes a mock testing platform that supports the simulation of microservice interfaces during both development and testing. The proposed approach enables the concurrent operation of real and simulated services, aiming to improve the efficiency, flexibility, and performance of microservice application development and testing.

Yu et al. [63] and Wu et al. [64] conduct API testing in MSA using *fault-based* strategies. More specifically, Yu et al. [63] introduces MicroRank, a method that analyzes both normal and abnormal traces to identify the root causes of latency issues in microservice environments. MicroRank extracts service latency from tracing data and performs anomaly detection. By combining PageRank with spectrum analysis, it ranks service instances responsible for latency issues, assigning them high scores. Wu et al. [64] presents MicroRCA, a system to locate root causes of performance issues in microservices. The root causes are inferred in real time by correlating application performance symptoms with corresponding system resource utilization, without any application instrumentation. The root cause localization of MicroRCA is based on an attributed graph that models anomaly propagation across services and machines.

API testing in MSA can also be approached from a *structure-based* perspective. Specifically, Abdelfattah et al. [62] address the challenge of assessing the thoroughness of MSA testing. To assist testers, the authors introduce test coverage metrics that evaluate the extent of end-to-end (E2E) test suite coverage for microservice endpoints. They also present an automated method for calculating the proposed metrics, providing feedback on the completeness of E2E test suites.

6. Non-functional testing

Non-functional testing ensures that MSAs meet critical quality attributes such as flexibility, performance, or reliability. The non-functional testing techniques proposed in the selected studies are classified in Table 2, providing a structured overview of their focus areas. The following sections present these techniques, organized according to the specific quality attributes they target.

6.1. Flexibility

While testing MSAs, flexibility is a factor that has been considered when assessing deployment alternatives or designing test strategies. Avritzer et al. [66] introduces an *experience-based* testing strategy, which proposes a quantitative approach for evaluating the scalability of MSAs under different workload scenarios. Their approach leverages operational profile data and automated testing to systematically assess deployment configurations. By incorporating input domain partitioning and domain-based load testing, Avritzer et al. [66] enable the evaluation of MSAs in terms of their suitability to handle specific operational workload conditions, providing a structured assessment of scalability.

Flexibility testing of MSAs can also happen through *usage-based* testing strategies, like the one proposed by Schulz et al. [65]. The *usage-based* testing strategy by Schulz et al. [65] emphasizes flexibility by tailoring load test workloads to target specific microservices and their dependencies rather than the entire system. The proposed algorithms – log-based and model-based – allow development teams to test individual services in isolation, considerably reducing resource consumption. While this tailored approach slightly reduces representativeness when compared to full system-level workload models, it enhances testing flexibility by limiting the number of microservices that must be deployed for testing.

6.2. Performance efficiency

Performance efficiency of MSAs is mostly tested via *specification-based* testing strategies, which are adopted by 15 primary studies. Out of these 15 studies, two papers focus on topics related to code quality, namely the correlation between architectural smells and performance deficiencies [74] and the impact of anti-patterns on performance [76]. More specifically, the study on architectural smells by Liu et al. [74] enables testing how architectural smells can cause performance degradation in MSAs. The study on anti-patterns by Matar and Jahić [76] builds on the results of Liu et al. [74] by presenting an approach to identify anti-patterns that affect performance in MSAs. The approach first uses static analysis to detect anti-patterns potentially affecting performance, and subsequently evaluates such assumption via a follow-up dynamic analysis.

Giamattei et al. [72] instead propose CAR-PT (Causal-Reasoning-driven Performance Testing), a model-based testing technique designed for the performance testing of MSA. CAR-PT leverages causal reasoning to guide the generation of workload configurations, enabling the identification of performance issues more effectively and efficiently. By automatically discovering and utilizing causal relationships between performance metrics (e.g., response time, CPU usage, and memory consumption), CAR-PT allows testers to simulate realistic workloads that target critical performance conditions.

Two other studies considering specification-based testing strategies focus on assessing the performance of software-intensive systems while migrating from a monolithic to an MSA [30,78]. Janes and Russo [30] propose a specification-based testing tool that enables to monitor the potential degradation of microservice performance based on the initial system specifications collected empirically via reference operational profiles. Maulana and Raharja [78] instead conduct a case study to assess the performance impact – and potential degradation – of transitioning a banking system from a monolithic to an MSA. In this case, potentially due to the practical nature of the study, the authors combine both load testing and stress testing strategies to assess the performance of the MSAs.

Microservice evolution is also considered by De Camargo et al. [96], who propose a framework to automate performance testing of microservice architectures. The framework, designed to be integrated in a continuous development environment, enables to automatically execute tests and collect related results by including a specification in each microservice. Based on such specification, test are automatically executed after each change of a microservice, allowing to monitor the performance of each microservice as they evolve in time.

At a higher level of abstraction, five studies considering specification-based testing strategies to propose approaches to evaluate the performance of MSAs [34,67,69,77,79]. The work of Matar and Jahic [77] presents a lightweight approach to assess the performance of early architectural ideas, e.g., to evaluate the impact that different architectural solutions may have on performance, or identify performance anti-patterns in the early design stages. This is achieved via an initial specification of the system, provided with a high-level architectural model created manually. The model is then used to statically evaluate its theoretical performance, generate the microservice code, and elaborate a test plan. As final step, performance testing results are collected dynamically, leading to a final performance report used for decision making. Similarly, Camilli et al. [67] present a declarative approach based on a custom domain specific language through which both standard performance testing approaches (e.g., load tests) and more advanced ones (e.g., stability boundary detection, and configuration tests) can be specified starting from templates. The approach is designed by considering its integration in continuous software development processes, allowing to monitor the fulfillment of specific performance intents throughout the evolution of MSAs.

By considering a more formal modeling of MSA performance, Smith et al. [34] propose a specification-based approach to reconstruct the performance behavior of MSAs by leveraging discrete time Markov chains. The presented approach is based on three subsequent phases, namely (i) sampling performance data in production to generate a coarse formal description of the users' behavior, (ii) conduct multiple load testing sessions to refine the model, and (iii) use the final specification created to evaluate different deployment options.

The two studies by Smith et al. [34] and Peuster et al. [79] instead focus on combining specification-based testing with monitoring capabilities. Specifically, Smith et al. [34] present a benchmark that can be used to study the efficacy and effectiveness of novel testing approaches, e.g., to study functional regression testing or load testing. The work of Peuster et al. [79] takes a more proactive stance, by presenting an approach to solution for virtualized, profile and jointly test microservice-based network functions and services. While considering microservice testing, the approach by Peuster et al. [79] is strongly tight to networking. Specifically, the authors present a solution based on the Testing and Test Control Notation, a testing language used in conformance testing of communicating systems, which enables to create a generic test framework for virtualized network functions implemented as microservices.

The last cluster of papers using specification-based testing approaches designed to study microservice performance focus on debugging and resolving performance issues [70,71,73,75]. More specifically, in the work of Lin et al. [73], an instrumentation-free approach is presented to identify performance issues, and potentially also perform root-cause analysis, in MSAs. The approach, named Microscope, reconstructs the causal graph of a MSA based on network connection information, and infer the causes of performance issue by tracing back anomalies detected in front-end service. Two related approaches make use of data-driven techniques to conduct performance debugging processes based on specification-based testing techniques. In the work of Gan et al. [71] a big data strategy is presented to detect quality of service violations based on spatial and temporal patterns. Specifically, the approach uses a distributed tracing system to records per-microservice latencies and number of requests queued per microservices, enabling to, which are then used to train a deep learning model to predict quality of service violations. In a similar follow up study, Gan et al. [70] focus instead on root cause analysis rather than debugging. In this case, an unsupervised model is trained on data collected from tracing systems of cloud providers, and subsequently leveraging latency the calculation of propagation and inter-service dependencies calculation to identify performance issues root causes. Finally, Lu [75] propose a dynamic approach to identify performance bottlenecks via stress testing, and subsequently resolve the identified issues by optimizing the deployment of MSAs.

From a different perspective, performance testing of MSAs can happen via other different testing strategies, such as *structure-based*, *usage-based*, and *experience-based* testing. A *structure-based* testing strategy adopted by Pei et al. [81], proposes a performance analysis approach for MSAs that leverages a multi-objective, search-based profiling technique. On the other hand, Camilli et al. [80] introduce an actor-driven decomposition methodology to enhance the modularization of MSAs while ensuring performance and scalability improvements.

Camilli et al. [82], Cooper et al. [37] and Jindal et al. [83] instead rely on *usage-based* strategies to test the performance efficiency of MSAs. Camilli et al. [82] introduce a methodology and support platform to automatically test MSA operations, which aims to be integrated into the DevOps cycle to support continuous testing. Cooper et al. [37] instead propose a framework that utilizes real-world usage traces to identify a minimal but essential set of performance tests. On the other hand, Jindal et al. [83] propose a modeling approach for testing and estimating the service capacity of individual microservices.

Last, but not least, Avritzer et al. [66] proposes an *experience-based* strategy for the performance assessment of MSAs deployment alternatives. Their approach uses automated performance testing results to quantitatively assess each deployment based on a proposed domain-based metric.

6.3. Reliability

As often happens in software systems, reliability testing of MSAs is currently mainly supported by means of *fault-based* testing strategies. Most of the existing works focus on system testing of MSAs [38,89,91–95,97]. Alvaro et al. [89] reasons backwards from working MSAs to determine whether failures in their execution could have prevented their outcomes [98]. Chen et al. [91] rely on non-intrusive request-level fault injection to prioritize high-impact failure test cases. Yang et al. [95] profile MSAs' resilience by correlating system failures with the degradation of user-experienced observed while injecting failures in the system. He et al. [38] proactively detects erroneous software changes reducing fault tolerance using fault injection and KPI analysis. Other frameworks leverage communication patterns for non-intrusive fault injections [94] or infer fault-tolerance bottlenecks to test redundancy without prior system knowledge [93].

Different, yet fault-based, reliability testing strategies are adopted by Meiklejohn et al. [92] and Assad et al. [90], who focus on integration testing. Meiklejohn et al. [92] enable the early testing of the resilience of service-to-service communications in an MSA under development. This is done by injecting failures in remote service calls, relying on the services' code to be instrumented to enable altering the response of such calls. Meiklejohn et al. [92] extends the above described solution by enabling the injection of Byzantine failures in service-to-database interactions.

Reliability testing of MSAs can also happen through *specification-based* testing strategies, like those proposed by Giamattei et al. [84,85]. The *specification-based* testing strategies by Giamattei et al. [84,85] rely on stateless pairwise combinatorial techniques for generating test cases for system testing of MSAs. Giamattei et al. [85] also provides the necessary tooling for executing and monitoring test results, as well as for identifying causal relations among the failures observed while testing a target MSA.

Last, but not least, Camilli et al. [82] and Pietrantuono et al. [87,88] propose *usage-based* strategies for reliability testing of MSAs, still at system level. More precisely, Camilli et al. [82] propose a methodology and support platform to enact ex-vivo testing sessions, i.e., in-house estimation of the reliability of a target MSA based on usage data monitored in prior, in-production runs. Pietrantuono et al. [87,88] instead support in-vivo testing by proposing two adaptive sampling schemes to identify test cases for an MSA based on its usage profile.

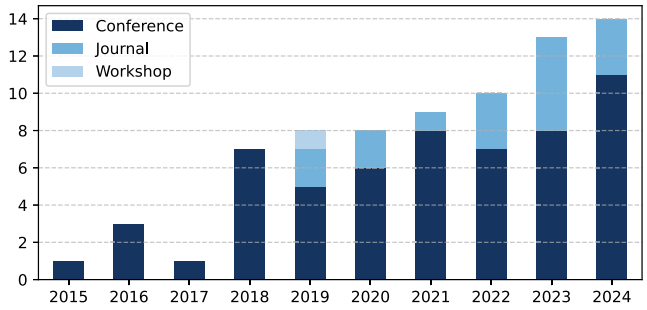


Fig. 2. Publication frequency per year.

7. Discussion

This section presents the key findings of our systematic review using 4W1H (i.e., *what*, *when*, *where*, *why*, and *how*) to comprehensively examine the landscape of testing MSAs. Please note that the terms in the categories can occur multiple times across different studies, as reported in Tables 1 and 2.

7.1. When and where: Publication trends

In order to understand the microservice testing research landscape, we first focus on the publication intensity of the topic throughout the years. An overview of the distribution of our primary studies throughout years, grouped by venue type, is documented in Fig. 2.

As we can observe from the figure, the topic of testing microservices first appeared in the literature in 2015. For a few years (2015–2017), few researchers considered publishing it. However, 2018 marked the start of a wave of popularity of the theme, with seven papers published. The topic never ceased to gain traction within the research community, with up to 14 papers published in a single year in 2024.

By considering the venues where studies on testing microservices are published, we can observe a recurrent trend. Specifically, towards their inception, research topics are first explored in workshops and conference venues, and only after getting consolidated start appearing in journal venues [99–101]. This trend can be observed also for microservice testing, with the topic appearing first in conferences (2015) and only after five years in journals. We conjecture that this trend reflects the rapid popularization that microservice literature experienced in 2015 [102], which after some years expanded also to the well-established software testing domain [103].

In terms of academic publishers, the majority of primary studies results to be released by IEEE (35 out of 74 primary studies), followed by ACM (16 studies), Springer (11 studies), and Elselvier (7 studies). The five remaining studies are published in Wiley (3 studies) and MDPI (2 studies).

Highlights

- Research on microservice testing has gained increasing momentum since 2015, peaking in 2024 with 14 publications—indicating growing interest and maturity in the field.

7.2. Why: Testing levels and objectives

To further explore the landscape of microservice testing research, we now examine the distribution of primary studies across different test levels and objectives.

Fig. 3 displays the distribution of primary studies across the different test levels as defined in the SWEBOK [9]: unit, integration, system, and acceptance testing. The distribution reveals a strong emphasis on system testing, which accounts for the largest group (47 studies). Given the complex interactions and non-functional requirements of microservices-based systems, system testing appears to be an ideal approach for evaluating overall behavior and performance. Integration testing follows with 21 studies, reflecting the importance of verifying communication between services. Unit testing is less emphasized than integration testing, with 15 studies. And acceptance testing is the least investigated (5 studies), suggesting limited attention to end-user validation.

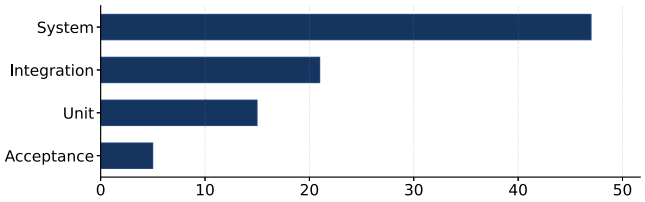


Fig. 3. Distribution of primary studies across the different Test Levels.

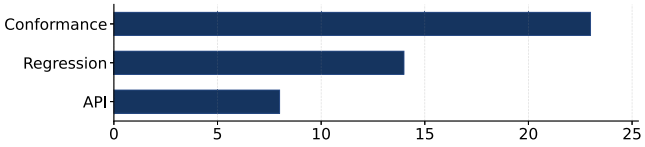


Fig. 4. Coverage of functional testing objectives.

We also categorized the primary studies based on testing objectives, distinguishing between functional and non-functional testing. A total of 44 studies focused on functional testing, while 36 studies concentrated on non-functional testing.

For the testing objectives, we refer to those listed in the SWEBOK [9]: conformance, compliance, installation, alpha and beta, regression, prioritization, non-functional, security, privacy, interface and application program interface (API), configuration, usability, and human–computer interaction.

Fig. 4 displays the distribution of primary studies across the different objectives. The studies were classified into three key testing objectives: Conformance, Regression, and API. Conformance testing is the most studied, with 23 primary studies, emphasizing the importance of ensuring that software systems adhere to specified standards, rules, and requirements. Regression testing follows with 14 studies, reflecting its critical role in verifying that modifications do not introduce unintended effects. API testing, with 8 studies, highlights the research efforts to ensure reliable interactions between system components. To offer a different perspective, Fig. 5 presents a combined view of the test levels and functional objectives addressed in the primary studies that focus on functional testing.

The studies focusing on non-functional testing were further categorized according to the nine quality characteristics outlined in the product quality model defined by ISO/IEC 25010 [15]: functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety. Fig. 6 displays the distribution of primary studies across the different quality characteristics.

Performance Efficiency is the most investigated characteristic, with 21 primary studies. These studies primarily focused on aspects such as response time, throughput, and resource utilization. Reliability follows with 14 studies investigating fault tolerance, availability, and recoverability in microservice architectures. In contrast, Flexibility is the least studied, with only 2 studies, highlighting the need for further investigation into flexibility in microservice testing to better support evolving software environments. Fig. 7 presents a combined view of

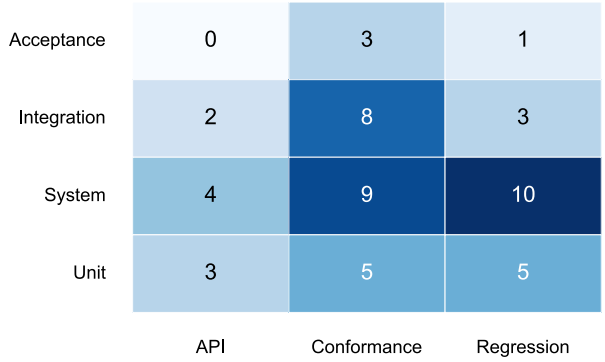


Fig. 5. Combined coverage of test level and functional testing objectives.

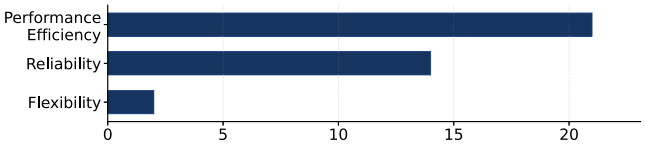


Fig. 6. Coverage of quality attributes.

Fig. 7. Combined coverage of test level and quality attributes.

the test levels and quality attributes addressed in the primary studies focusing on non-functional testing.

Highlights

- System testing is the most commonly studied level (47 studies), highlighting its importance in evaluating complex behaviors and non-functional requirements of MSAs.
- While performance efficiency and reliability dominate non-functional testing, quality attributes such as flexibility, maintainability, and security remain significantly underexplored, indicating potential gaps and research directions.

7.3. How: Study types and testing strategies

As illustrated in Fig. 8, *specification-based* emerges as the most widely used strategy. This indicates that the majority of the proposals focus on selecting a subset of test cases from the input domain to detect specific categories of faults. When this insight is combined with the data presented in Fig. 9, we observe that specification-based strategies

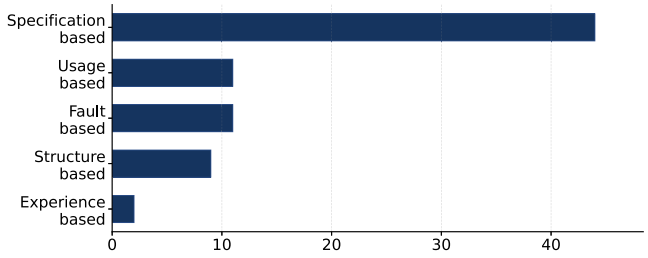


Fig. 8. Coverage of testing strategies.

Fig. 9. Combined coverage of testing strategies and functional objectives.

are predominantly applied in conformance and regression testing. Conformance testing aims to ensure that the SUT adheres to predefined standards, rules, specifications, and requirements. Given its structured approach, *specification-based* strategies are used to systematically test the MSA against formally defined fault categories. Similarly, regression testing, which involves the selective retesting of a SUT to verify that modifications have not introduced unintended side effects, also heavily relies on specification-based strategies. This reinforces the role of specification-based strategies in maintaining the MSA compliance and integrity.

When analyzing the relation between testing strategies and quality attributes (Fig. 10), we observe that *specification-based* is once again the most widely used strategy. In this context, it is primarily applied to *Performance Efficiency*, with a lesser emphasis on *Reliability*. In contrast, proposals using *fault-based* strategies are exclusively focused on *Reliability*, targeting the detection of faults that could, perhaps, compromise the MSA availability. On the other hand, *usage-based* strategies are applied in a more distributed manner, addressing *Performance Efficiency*, *Reliability*, and *Flexibility*. This suggests that the usage-based strategy offers a broader perspective by considering different aspects of the MSA quality.

Regarding study types, as shown in Fig. 11, there is a clear preference for controlled environments. The most common study type is Laboratory Experiment (50, 63%), followed by Formal Theory (12, 15%) and Computer Simulation (7, 8%). In contrast, Field, Judgment, and Sample Studies are rarely conducted, highlighting the limited empirical validation of testing approaches. This reliance on laboratory experiments suggests that most findings are derived from structured, artificial settings. While these settings are valuable for isolating variables and ensuring repeatability, they may fail to capture the complexities of real-world operational environments. Similarly, most primary studies use black-box testing, and only three incorporate gray-box or white-box approaches.

From the perspective of Technology Readiness Levels (TRL), most proposals remain in the low to mid TRL range (TRL 3–4), meaning they are still in the experimental or proof-of-concept stages. To bridge the

Fig. 10. Combined coverage of testing strategies and quality attributes.

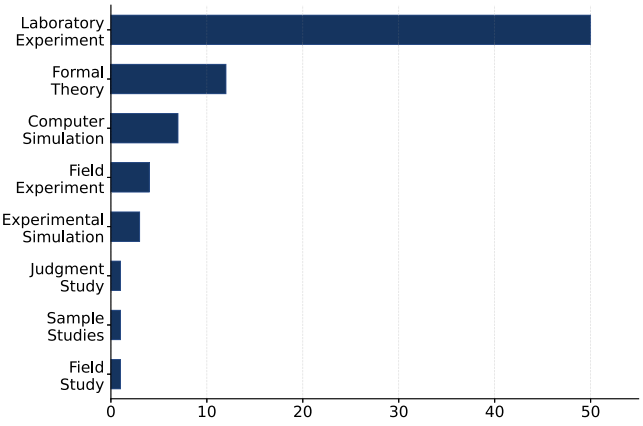


Fig. 11. Coverage of study types.

gap between research and industry adoption, future studies should focus on achieving higher TRL, integrating field studies and experiments to showcase the effectiveness of these approaches in complex MSAs.

Highlights

- Specification-based is the most used strategy, primarily applied in conformance and regression testing to ensure MSA compliance and integrity.
- Most proposals rely on laboratory experiments, limiting real-world validation; Increasing the current TRL is crucial for industry adoption.

7.4. What: SUTs

Another interesting dimension in analyzing the state-of-the-art of microservice testing lies in the *what*, namely in the size and nature of the Systems Under Test (SUTs) employed in the selected studies. Given the inherently distributed and modular architecture of MSAs, the size of SUTs can influence testing strategies, tooling efficacy, and overall evaluation outcomes. To enable a first structured comparison of the SUTs covered in the selected studies, we first classified them based on the number of constituent microservices: *small* MSAs comprise up to 9 microservices, *medium* MSAs include 10 to 20 microservices, and *large* MSAs consist of at least 21 microservices.

Notably, all SUTs sizes result to be significantly covered in the selected studies, with *small* SUTs used in 21 studies, *medium* SUTs used in 14 studies, and *large* SUTs used in 22 studies. As for *small* SUTs, this mainly depends on the fact that small demo MSAs are used to

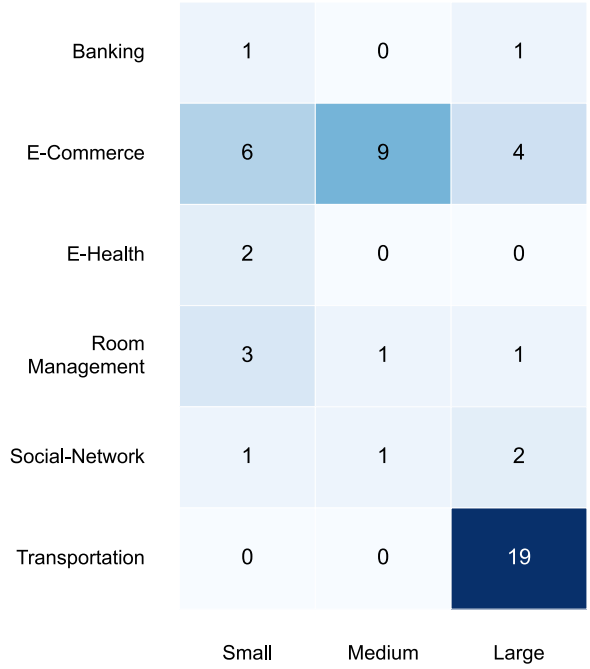


Fig. 12. Combined coverage of SUTs' application domains and sizes (with general purpose demo MSAs excluded from the count).

experiment with the microservice testing techniques proposed in 21 of the selected studies, e.g., [37,40,83,88,92]. This widespread reliance on small MSAs underscores the need for lightweight MSAs to be used as a common and manageable basis for the rapid evaluation of microservice testing approaches. However, no existing small MSA seems to emerge as widely accepted or used in the selected studies. This calls for small MSAs to be identified/proposed to serve as a standard benchmark for the rapid evaluation of newly proposed microservice testing solutions.

Different arguments instead apply to *medium* and *large* SUTs, which are typically used to provide a more thorough assessment of the microservice testing solutions proposed by 22 of the selected studies. As shown in Fig. 12, almost all the selected studies using medium or large SUTs relied on MSAs coming from the e-commerce or transportation application domains, respectively. This is mainly motivated by the fact that SockShop.[3] and TrainTicket[4] emerge as the de-facto reference medium and large SUTs. Sock Shop and TrainTicket are indeed the most used medium and large SUTs, with Sock Shop used to assess what is proposed by 6 selected studies [64,65,67,73,84,93], while TrainTicket is used in 16 selected studies [25,26,29,34,36,44,53,62,74,80–82,84–86,91,93,95]. This is in line with what is observed by Fischer et al. [104], however, with the additional bit that Sock Shop is now marked as deprecated and no longer maintained. This raises the issue of answering the need for a medium-sized MSA to be used as a reference SUT for newly proposed microservice testing solutions.

Highlights

- Need for standardized *small* and *medium* MSAs to be used as manageable SUTs.
- TrainTicket is the de-facto standard when needing a large MSA to be used as SUT to assess proposals thoroughly.

[3] https://github.com/microservices-demo/microservices-demo
[4] https://github.com/FudanSELab/train-ticket

8. Threats to validity

In this section, we discuss the threats to the validity of our study. Following the categorization proposed by Wohlin et al. [105], and the guidelines proposed by Ampatzoglou et al. [106], we address four categories of threats that may affect the validity of our findings.

8.1. External validity

The potential threats to the external validity of a study affect the applicability of the study's results in a broader and more general context [105]. One potential threat to the validity of our results arises from the selection of primary studies used to achieve our research objective. To mitigate this, we employed a comprehensive digital literature indexer (namely Google Scholar), which aggregates publications from various major digital libraries, such as Scopus and Web of Science. To further reduce bias and ensure coverage, we applied a recursive, bidirectional snowballing process, continuing the search until theoretical saturation was reached. Another threat to external validity arises from the nature of the primary studies themselves, which are limited to peer-reviewed academic literature. The scope of our review was deliberately defined prior to execution and explicitly excludes white and grey literature. As such, we emphasize that our objective is to provide an overview of the state of the art in microservices testing. In future work, we plan to complement our results with an analysis of the state of practice based on white and grey literature available on the topic.

8.2. Internal validity

The internal validity of a study concerns the validity of the method employed to analyze the study data [105]. An internal threat to validity inherent in our chosen research method, namely an SLR, concerns the soundness of the research steps and the rigor of their execution. To mitigate such threats, we adhered to established and widely recognized guidelines for conducting SLRs and snowballing procedures [8,106,107]. To address potential biases related to subjective interpretation of the selected literature, we implemented several mitigation strategies. First, we defined a clear set of selection criteria and data extraction attributes prior to conducting the review. This structured research framework was consistently applied throughout the study, with any uncertainties or ambiguities resolved through thorough discussion among all researchers. Second, we executed a supplementary automated query to search for potentially primary studies on a second source of data, namely the Scopus digital library. Third, regular weekly meetings were held throughout the review process to ensure ongoing alignment in literature selection and data extraction decisions.

As outlined by Gusenbauer and Haddaway [108] and Piasecki et al. [109] the use of Google Scholar constitutes a potential threat to reproducibility of the study. However, as described by Wholin in his seminar work "Guidelines for snowballing in systematic literature studies and a replication in software engineering" [8] the search engine can constitute a source of data to initiate a snowballing process, as done in this research. Therefore, on one hand we mitigate potential threats entailed by utilizing Google scholar by conducting a thorough bidirectional snowballing process adopting theoretical saturation as stop criterion. On the other hand, we promote the independent replication and scrutiny of our work through a set of *a priori* detailed research steps and a rigorous replication package including the totality of the intermediate and final research outputs and traces. To further mitigate potential threats related to utilizing Google Scholar, we adopted also a supplementary automated query executed on an additional data source, namely the widely-adopted digital library Scopus, which provided results coherent with those obtained from Google Scholar (Section 4.2).

8.3. Construct and conclusions validity

Construct and conclusion validity relate to the generalizability of the constructs under investigation and the extent to which the study's conclusions are reasonably supported by the available data, respectively [105]. In our SLR, these aspects may be influenced by observer bias and interpretive subjectivity. To mitigate such threats, we quantitatively evaluated inter-rater agreement using Fleiss' Kappa, which indicated a substantial level of agreement among raters (see Section 4.4). In addition, conclusions were independently derived by multiple researchers and subsequently reconciled through collaborative discussion, during which interpretations were cross-checked against the selected studies and related literature. As an additional mitigation strategy, all intermediate and final artifacts produced during this research, including the paper selection processes, extracted data, and source code utilized, are made publicly available for the sake of scrutiny and replicability. Finally, our research procedures are thoroughly documented and grounded in established, widely adopted methodological guidelines [8,106,107], ensuring transparency and helping to identify any remaining, unrecognized threats to construct and conclusion validity.

9. Conclusion

While MSAs offer substantial benefits in terms of scalability, fault isolation, and continuous deployment, their distributed and heterogeneous nature introduces significant testing complexities. To address the fragmented and diverse landscape of microservice testing research, this SLR consolidated and classified 74 primary studies, offering an overview of current practices and research trends.

Based on the information collected from the primary studies described in Sections 5 and 6, and the key findings discussed in Section 7, we hereafter describe the practical implications of our results (Section 9.1) and possible future directions for advancing microservice testing research (Section 9.2).

9.1. Practical implications

The classification provided in this SLR can serve as a first support for practitioners facing the fragmented landscape of microservice testing. Practitioners can consult our classification to identify testing approaches best aligned with their needs and architectural configurations.

For example, organizations working on new MSAs can benefit from specification-based functional testing early in the development lifecycle. These approaches ensure API conformance, enforce contract-based interactions, and detect integration issues before deployment [59–61].

For organizations decomposing a monolithic into an MSA, instead, system-level regression testing becomes critical to ensure that decomposed services preserve functional behavior. Approaches such as microRTS [28], CIPC [26], and symbolic test mapping [36] support regression tracking, while tools like PPTAM+ [30] and performance tracing can help to monitor performance degradation across versions [30,78].

As MSA scale, development teams may prioritize non-functional testing strategies. Our SLR highlights relevant techniques for performance and reliability evaluation, which are mainly achieved through specification-based and fault-based testing strategies, respectively. Development teams could combine performance efficiency techniques, like the one proposed by Matar and Jahic [77], with fault-based reliability testing (e.g., He et al. [38]). On the other hand, runtime-informed test strategies, such as budget-aware test selection [37], ex-vivo testing based on production data [82], and change-based test prioritization [25], can help to optimize test coverage under time and resource constraints.

For teams with mature DevOps pipelines, understanding which testing strategies support rapid iteration without compromising safety is essential. The literature gathered during our SLR shows a strong emphasis

on automated, repeatable, and low-intrusion testing, typically achieved through specification-based and fault-based strategies. This suggests that automated, repeatable tests are central for production-ready MSAs.

9.2. Future research directions

Our SLR findings also provide useful insights for researchers, who in turn can leverage our results to explore potential new research directions, such as those described below:

- *Underexplored Quality Attributes*: Although performance and reliability have received substantial attention from the collected primary studies, quality attributes such as maintainability, flexibility, and security remain notably underexplored. Future research should explore how to evaluate these attributes in dynamic and evolving MSAs.
- *Security-oriented Testing Strategies*: Among the underexplored quality attributes, it is astonishing that none of the selected primary studies directly address security testing on MSAs. This represents a substantial research gap. Future research could explore how to test MSAs for security vulnerabilities, such as improper service exposure, insufficient authentication or authorization, and insecure service-to-service communication [110].
- *Empirical Validation*: The predominance of studies at TRL 3–4 highlights a research gap for the applicability of the proposed approaches. Future research should prioritize evaluating testing approaches in industrial contexts to expose real-world constraints and challenges. Collaborations with industry partners, coupled with the development of standardized testbeds and datasets, will enhance reproducibility and support community-driven benchmarking.
- *Standardized Small and Medium MSAs*: There is a lack of standardized, representative small and medium-scale MSAs for microservice testing. Future research could focus on developing a suite of medium-complexity MSAs, including diverse domain logic and varied inter-service dependencies. This would provide a bridge between the current overly simplistic toy examples and the default standard large MSA (i.e., TrainTicket).
- *Unexplored Testing Dimensions*: Some functional testing dimensions remain unexplored for microservices. For example, compliance testing, which verifies adherence to external regulations like data protection laws or industry-specific standards, is critical, especially in domains such as finance and healthcare. Additionally, usability and human–computer interaction testing were not addressed by the primary studies. This is a notable oversight given their growing relevance as microservices increasingly underpin frontend experiences. Future research could explore testing techniques that assess user learnability, efficiency, and resilience to errors.

CRediT authorship contribution statement

Francisco Ponce: Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Data curation, Conceptualization. **Roberto Verdecchia:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Data curation. **Breno Miranda:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Data curation, Conceptualization. **Jacopo Soldani:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partly supported by the project *FREEDA* (CUP: I53D23003550006), funded by the frameworks PRIN (MUR, Italy) and Next Generation EU.

Data availability

A replication package for our study is available at https://doi.org/10.5281/zenodo.16941552.

References

[1] Nane Kratzke, Peter-Christian Quint, Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study, J. Syst. Softw. 126 (2017) 1–16, http://dx.doi.org/10.1016/j.jss.2017.01.001.
[2] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen, Microservice architecture: Aligning principles, practices, and culture, first ed., O'Reilly Media, Inc., 2016.
[3] Amr S. Abdelfattah, Tomas Cerny, Roadmap to reasoning in microservice systems: A rapid review, Appl. Sci. 13 (3) (2023) http://dx.doi.org/10.3390/app13031838.
[4] Mikko Pirhonen, Kari Systä, David Hästbacka, The pains and gains of microservices revisited, in: Product-Focused Software Process Improvement: 25th International Conference, PROFES 2024, Tartu, Estonia, December 2–4, 2024, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2024, pp. 238–254, http://dx.doi.org/10.1007/978-3-031-78386-9_16.
[5] Jacopo Soldani, Damian Andrew Tamburri, Willem-Jan Van Den Heuvel, The pains and gains of microservices: A systematic grey literature review, J. Syst. Softw. 146 (2018) 215–232, http://dx.doi.org/10.1016/j.jss.2018.09.082.
[6] Tomas Cerny, Amr S. Abdelfattah, Vincent Bushong, Abdullah Al Maruf, Davide Taibi, Microservice architecture reconstruction and visualization techniques: A review, in: 2022 IEEE International Conference on Service-Oriented System Engineering, SOSE, 2022, pp. 39–48, http://dx.doi.org/10.1109/SOSE55356.2022.00011.
[7] Barbara Kitchenham, Procedures for performing systematic reviews, Keele, UK, Keele Univ. 33 (2004) (2004) 1–26.
[8] Claes Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: International Conference on Evaluation and Assessment in Software Engineering, ACM Press, 2014, pp. 1–10.
[9] Hironori Washizaki, Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0, IEEE Computer Society, Waseda University, Japan, 2024, http://www.swebok.org.
[10] James Lewis, Martin Fowler, Microservices: a definition of this new architectural term, MartinFowler.Com 25 (14–26) (2014) 12.
[11] Sam Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, Inc., 2021.
[12] Olaf Zimmermann, Microservices tenets: Agile approach to service development and deployment, Comput. Science-Research Dev. 32 (3) (2017) 301–310.
[13] Claus Pahl, Antonio Brogi, Jacopo Soldani, Pooyan Jamshidi, Cloud container technologies: a state-of-the-art review, IEEE Trans. Cloud Comput. 7 (3) (2017) 677–692.
[14] Thomas C. Fountain, Web service oriented architecture:" smart operations" and it strategy., in: ICWS, 2003, pp. 481–486.
[15] International Organization For Standardization, ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, in: System and Software Quality Models, vol. 2, 2023.
[16] Stefan Fischer, Pirmin Urbanke, Rudolf Ramler, Monika Steidl, Michael Felderer, An overview of microservice-based systems used for evaluation in testing and monitoring: a systematic mapping study, in: Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024), 2024, pp. 182–192.
[17] Israr Ghani, Wan MN Wan-Kadir, Ahmad Mustafa, Muhammad Imran Babir, Microservice testing approaches: A systematic literature review, Int. J. Integr. Eng. 11 (8) (2019) 65–80.
[18] Mingxuan Hui, Lu Wang, Hao Li, Ren Yang, Yuxin Song, Huiying Zhuang, Di Cui, Qingshan Li, Unveiling the microservices testing methods, challenges, solutions, and solutions gaps: A systematic mapping study, J. Syst. Softw. (2024) 112232.
[19] Manuel Simosa, Frank Siqueira, Contract testing in microservices-based systems: A survey, in: 2023 IEEE 14th International Conference on Software Engineering and Service Science, ICSESS, IEEE, 2023, pp. 312–317.
[20] Muhammad Waseem, Peng Liang, Gastón Márquez, Amleto Di Salle, Testing microservices architecture-based applications: A systematic mapping study, in: 2020 27th Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2020, pp. 119–128.

[21] Victor R. Basili, Gianluigi Caldiera, Dieter Rombach, The goal question metric approach, in: Encyclopedia of Software Engineering, Wiley, 1994, pp. 528–532.

[22] Neal R. Haddaway, Matthew J. Page, Chris C. Pritchard, Luke A. McGuinness, PRISMA2020: An R package and shiny app for producing PRISMA 20202010compliant flow diagrams, with interactivity for optimised digital transparency and open synthesis, Campbell Syst. Rev. 18 (2) (2022) http://dx.doi.org/10.1002/cl2.1230.

[23] Johnny Saldaña, The coding manual for qualitative researchers, third ed., SAGE Publications, Los Angeles, 2021.

[24] Klaas-Jan Stol, Brian Fitzgerald, The ABC of software engineering research, ACM Trans. Softw. Eng. Methodol. (TOSEM) 27 (3) (2018) 1–51.

[25] Lizhe Chen, Ji Wu, Haiyan Yang, Kui Zhang, Does PageRank apply to service ranking in microservice regression testing? Softw. Qual. J. 30 (3) (2022) 757–779.

[26] Lizhe Chen, Xiang Yu, Ji Wu, Haiyan Yang, CIPC: A change impact propagation computing based technique for microservice regression testing prioritization, Mob. Inf. Syst. 2021 (1) (2021) 2912240.

[27] Emanuele De Angelis, Guglielmo De Angelis, Alessandro Pellegrini, Maurizio Proietti, Inferring relations among test programs in microservices applications, in: 2021 IEEE International Conference on Service-Oriented System Engineering, SOSE, IEEE, 2021, pp. 114–123.

[28] Daniel Elsner, Daniel Bertagnolli, Alexander Pretschner, Rudi Klaus, Challenges in regression test selection for end-to-end testing of microservice-based software systems, in: Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test, 2022, pp. 1–5.

[29] Luca Gazzola, Maayan Goldstein, Leonardo Mariani, Marco Mobilio, Itai Segall, Alessandro Tundo, Luca Ussi, ExVivoMicroTest: ExVivo testing of microservices, J. Softw.: Evol. Process. 35 (4) (2023) e2452.

[30] Andrea Janes, Barbara Russo, Automatic performance monitoring and regression testing during the transition from monolith to microservices, in: 2019 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, IEEE, 2019, pp. 163–168.

[31] Mohammad Javad Kargar, Alireza Hanifizade, Automation of regression test in microservice architecture, in: 2018 4th International Conference on Web Research, ICWR, IEEE, 2018, pp. 133–137.

[32] Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, I-Hsiu Liu, Ci-Wei Lan, Graph-based and scenario-driven microservice analysis, retrieval, and testing, Future Gener. Comput. Syst. 100 (2019) 724–735.

[33] Michael Schneider, Stephanie Zieschinski, Hristo Klechorov, Lukas Brosch, Patrick Schorsten, Sebastian Abeck, Christof Urbaczek, A test concept for the development of microservice-based applications, in: ICSEA 2021, 2021, p. 98.

[34] Sheldon Smith, Ethan Robinson, Timmy Frederiksen, Trae Stevens, Tomas Cerny, Miroslav Bures, Davide Taibi, Benchmarks for end-to-end microservices testing, in: 2023 IEEE International Conference on Service-Oriented System Engineering, SOSE, IEEE, 2023, pp. 60–66.

[35] Li-zhe Chen, Ji Wu, Hai-yan Yang, Kui Zhang, A microservice regression testing selection approach based on belief propagation, J. Cloud Comput. 12 (1) (2023) 20.

[36] Emanuele De Angelis, Guglielmo De Angelis, Alessandro Pellegrini, Maurizio Proietti, What makes test programs similar in microservices applications? J. Syst. Softw. 201 (2023) 111674, http://dx.doi.org/10.1016/j.jss.2023.111674, URL https://www.sciencedirect.com/science/article/pii/S0164121223000699.

[37] Quinn Cooper, Diwakar Krishnamurthy, Yasaman Amannejad, Budget aware performance test selection for microservices, in: 2024 IEEE 17th International Conference on Cloud Computing, CLOUD, 2024, pp. 376–385, http://dx.doi.org/10.1109/CLOUD62652.2024.00049.

[38] Guanglei He, Xiaohui Nie, Ruming Tang, Kun Wang, Zhaoyang Yu, Xidao Wen, Kanglin Yin, Dan Pei, Guardian of the resiliency: Detecting erroneous software changes before they make your microservice system less fault-resilient, in: 2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS), 2024, pp. 1–10, http://dx.doi.org/10.1109/IWQoS61813.2024.10682951.

[39] Hamdy Michael Ayas, Hartmut Fischer, Philipp Leitner, Francisco Gomes De Oliveira Neto, An empirical analysis of microservices systems using consumer-driven contract testing, in: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, IEEE, 2022, pp. 92–99, http://dx.doi.org/10.1109/SEAA56994.2022.00022.

[40] Huixia Ding, Lei Cheng, Qinyuan Li, An automatic test data generation method for microservice application, in: 2020 International Conference on Computer Engineering and Application, ICCEA, 2020, pp. 188–191, http://dx.doi.org/10.1109/ICCEA50009.2020.00048.

[41] Tingting Duan, Da Li, Jiaxing Xuan, Fanjie Du, Jing Li, Jing Du, Shang Wu, Design and implementation of intelligent automated testing of microservice application, in: 2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference, ITNEC, 5, 2021, pp. 1306–1309, http://dx.doi.org/10.1109/ITNEC52019.2021.9587260.

[42] Victor Heorhiadi, Shriram Rajagopalan, Hani Jamjoom, Michael K. Reiter, Vyas Sekar, Gremlin: Systematic resilience testing of microservices, in: 2016 IEEE 36th International Conference on Distributed Computing Systems, ICDCS, 2016, pp. 57–66, http://dx.doi.org/10.1109/ICDCS.2016.11.

[43] Na Li, Jun Wang, Chen Chen, Hongfei Hu, Application of API automation testing based on microservice mode in industry software, in: Proceedings of the International Conference on Algorithms, Software Engineering, and Network Security, ASENS '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 460–464, http://dx.doi.org/10.1145/3677182.3677264.

[44] Zhenyue Long, Guoquan Wu, Xiaojiang Chen, Chengxu Cui, Wei Chen, Jun Wei, Fitness-guided resilience testing of microservice-based applications, in: 2020 IEEE International Conference on Web Services, ICWS, 2020, pp. 151–158, http://dx.doi.org/10.1109/ICWS49710.2020.00027.

[45] Gang Luo, Xi Zheng, Huai Liu, Rongbin Xu, Dinesh Nagumothu, Ranjith Janapareddi, Er Zhuang, Xiao Liu, Verification of microservices using metamorphic testing, in: Algorithms and Architectures for Parallel Processing: 19th International Conference, ICA3PP 2019, Melbourne, VIC, Australia, December 9–11, 2019, Proceedings, Part I 19, Springer, 2020, pp. 138–152.

[46] Shang-Pin Ma, I-Hsiu Liu, Chun-Yu Chen, Yu-Te Wang, Version-based and risk-enabled testing, monitoring, and visualization of microservice systems, J. Softw.: Evol. Process. 34 (10) (2022) e2429, http://dx.doi.org/10.1002/smr.2429, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2429.

[47] Shang-Pin Ma, Yu-Yung Yang, Shin-Jie Lee, Hang-Wei Yeh, UTEMS: A unit testing scheme for event-driven microservices, in: 2023 10th International Conference on Dependable Systems and their Applications, DSA, 2023, pp. 591–592, http://dx.doi.org/10.1109/DSA59317.2023.00085.

[48] Mazedur Rahman, Jerry Gao, A reusable automated acceptance testing architecture for microservices in behavior-driven development, in: 2015 IEEE Symposium on Service-Oriented System Engineering, IEEE, 2015, pp. 321–325.

[49] Christoph Reile, Mohak Chadha, Valentin Hauner, Anshul Jindal, Benjamin Hofmann, Michael Gerndt, Bunk8s: Enabling easy integration testing of microservices in kubernetes, in: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER, 2022, pp. 459–463, http://dx.doi.org/10.1109/SANER53432.2022.00062.

[50] Yang Wang, Lei Cheng, Xin Sun, Design and research of microservice application automation testing framework, in: 2019 International Conference on Information Technology and Computer Application, ITCA, 2019, pp. 257–260, http://dx.doi.org/10.1109/ITCA49981.2019.00063.

[51] Chu-Fei Wu, Shang-Pin Ma, An-Chi Shau, Hang-Wei Yeh, Testing for event-driven microservices based on consumer-driven contracts and state models, in: 2022 29th Asia-Pacific Software Engineering Conference, APSEC, 2022, pp. 467–471, http://dx.doi.org/10.1109/APSEC57359.2022.00064.

[52] Shenglin Zhang, Jun Zhu, Bowen Hao, Yongqian Sun, Xiaohui Nie, Jingwen Zhu, Xilin Liu, Xiaoqian Li, Yuchi Ma, Dan Pei, Fault diagnosis for test alarms in microservices through multi-source data, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, in: FSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, pp. 115–125, http://dx.doi.org/10.1145/3663529.3663833.

[53] Amr S. Abdelfattah, Tomas Cerny, Jorge Yero, Eunjee Song, Davide Taibi, Test coverage in microservice systems: An automated approach to E2E and API test coverage metrics, Electronics 13 (10) (2024) http://dx.doi.org/10.3390/electronics13101913.

[54] Theofanis Vassiliou-Gioles, Quality assurance of micro-services - when to trust your micro-service test results? in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2021, pp. 01–06, http://dx.doi.org/10.1109/QRS-C55045.2021.00024.

[55] Chu-Fei Wu, Shang-Pin Ma, An-Chi Shau, Hang-Wei Yeh, Testing for event-driven microservices based on consumer-driven contracts and state models, in: 2022 29th Asia-Pacific Software Engineering Conference, APSEC, 2022, pp. 467–471, http://dx.doi.org/10.1109/APSEC57359.2022.00064.

[56] Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, Wen-Tin Lee, Shin-Jie Lee, Nien-Lin Hsueh, Using service dependency graph to analyze and test microservices, in: 2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC, vol. 02, 2018, pp. 81–86, http://dx.doi.org/10.1109/COMPSAC.2018.10207.

[57] Mustafa Almutawa, Qusai Ghabrah, Marco Canini, Towards LLM-assisted system testing for microservices, in: 2024 IEEE 44th International Conference on Distributed Computing Systems Workshops, ICDCSW, IEEE, 2024, pp. 29–34, http://dx.doi.org/10.1109/ICDCSW63686.2024.00011.

[58] Zhenhe Yao, Haowei Ye, Changhua Pei, Guang Cheng, Guangpei Wang, Zhiwei Liu, Hongwei Chen, Hang Cui, Zeyan Li, Jianhui Li, Gaogang Xie, Dan Pei, SparseRCA: Unsupervised root cause analysis in sparse microservice testing traces, in: 2024 IEEE 35th International Symposium on Software Reliability Engineering, ISSRE, 2024, pp. 391–402, http://dx.doi.org/10.1109/ISSRE62328.2024.00045.

[59] Pattarakrit Rattanukul, Chansida Makaranond, Pumipat Watanakulcharus, Chaiyong Ragkhitwetsagul, Tanapol Nearunchorn, Vasaka Visoottiviseth, Morakot Choetkiertikul, Thanwadee Sunetnanta, Microusity: A testing tool for backends for frontends (BFF) microservice systems, in: 2023 IEEE/ACM 31st International Conference on Program Comprehension, ICPC, IEEE, 2023, pp. 74–78.

[60] Nikita Ashikhmin, Gleb Radchenko, Andrei Tchernykh, RAML-based mock service generator for microservice applications testing, in: Supercomputing: Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25–26, 2017, Revised Selected Papers 3, Springer, 2017, pp. 456–467.

[61] Du Lin, Fang Liping, Han Jiajia, Tang Qingzhao, Sun Changhua, Zhang Xiaohui, Research on microservice application testing based on mock technology, in: 2020 International Conference on Virtual Reality and Intelligent Systems, ICVRIS, IEEE, Placeholder Address, 2020, pp. 815–819.

[62] Amr S. Abdelfattah, Tomas Cerny, Jorge Yero Salazar, Austin Lehman, Joshua Hunter, Ashley Bickham, Davide Taibi, End-to-end test coverage metrics in microservice systems: An automated approach, in: European Conference on Service-Oriented and Cloud Computing, Springer, One New York Plaza, Suite 4600, New York, 2023, pp. 35–51.

[63] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, Xiaoyun Li, Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments, in: Proceedings of the Web Conference 2021, 2021, pp. 3087–3098.

[64] Li Wu, Johan Tordsson, Erik Elmroth, Odej Kao, Microrca: Root cause localization of performance issues in microservices, in: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2020, pp. 1–9.

[65] Henning Schulz, Tobias Angerstein, Dušan Okanović, André van Hoorn, Microservice-tailored generation of session-based workload models for representative load testing, in: 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS, 2019, pp. 323–335, http://dx.doi.org/10.1109/MASCOTS.2019.00043.

[66] Alberto Avritzer, Vincenzo Ferme, Andrea Janes, Barbara Russo, André van Hoorn, Henning Schulz, Daniel Menasché, Vilc Rufino, Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests, J. Syst. Softw. 165 (2020) 110564, http://dx.doi.org/10.1016/j.jss.2020.110564, URL https://www.sciencedirect.com/science/article/pii/S016412122030042X.

[67] Matteo Camilli, Andrea Janes, Barbara Russo, Automated test-based learning and verification of performance models for microservices systems, J. Syst. Softw. 187 (2022) 111225.

[68] André de Camargo, Ivan Salvadori, Ronaldo dos Santos Mello, Frank Siqueira, An architecture to automate performance tests on microservices, iiWAS '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 422–429, http://dx.doi.org/10.1145/3011141.3011179.

[69] Vincenzo Ferme, Cesare Pautasso, A declarative approach for performance tests execution in continuous software development environments, in: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018, pp. 261–272.

[70] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, Christina Delimitrou, Sage: practical and scalable ML-driven performance debugging in microservices, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, pp. 135–151.

[71] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, Christina Delimitrou, Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 19–33.

[72] Luca Giamattei, Antonio Guerriero, Ivano Malavolta, Cristian Mascia, Roberto Pietrantuono, Stefano Russo, Identifying performance issues in microservice architectures through causal reasoning, in: Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024), 2024, pp. 149–153.

[73] JinJin Lin, Pengfei Chen, Zibin Zheng, Microscope: Pinpoint performance issues with causal graphs in micro-service environments, in: Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16, Springer, 2018, pp. 3–20.

[74] Lei Liu, Zhiying Tu, Xiang He, Xiaofei Xu, Zhongjie Wang, An empirical study on underlying correlations between runtime performance deficiencies and "bad smells" of microservice systems, in: 2021 IEEE International Conference on Web Services, ICWS, IEEE, 2021, pp. 751–757.

[75] Zhijie Lu, Research on performance optimization method of test management system based on microservices, in: 2023 4th International Conference on Computer Engineering and Application, ICCEA, IEEE, 2023, pp. 445–451.

[76] Raghad Matar, Jasmin Jahić, An approach for evaluating the potential impact of anti-patterns on microservices performance, in: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), IEEE, 2023, pp. 167–170.

[77] Raghad Matar, Jasmin Jahic, MDEPT: Microservices design evaluator and performance tester, in: European Conference on Software Architecture, Springer, 2024, pp. 138–154.

[78] Alwi Maulana, Pradana Ananda Raharja, Design and testing on migration of remiss-supply in banking system to microservice architecture, in: 2022 IEEE International Conference on Communication, Networks and Satellite, COMNETSAT, IEEE, 2022, pp. 168–173.

[79] Manuel Peuster, Christian Dröge, Clemens Boos, Holger Karl, Joint testing and profiling of microservice-based network services using TTCN-3, ICT Express 5 (2) (2019) 150–153.

[80] Matteo Camilli, Carmine Colarusso, Barbara Russo, Eugenio Zimeo, Actor-driven decomposition of microservices through multi-level scalability assessment, ACM Trans. Softw. Eng. Methodol. 32 (5) (2023) http://dx.doi.org/10.1145/3583563.

[81] Qiugen Pei, Zheheng Liang, Zeling Wang, Lei Cui, Zhenyue Long, Guoquan Wu, Search-based performance testing and analysis for microservice-based digital power applications, in: 2023 6th International Conference on Energy, Electrical and Power Engineering, CEEPE, 2023, pp. 1522–1527, http://dx.doi.org/10.1109/CEEPE58418.2023.10165808.

[82] Matteo Camilli, Antonio Guerriero, Andrea Janes, Barbara Russo, Stefano Russo, Microservices integrated performance and reliability testing, in: 2022 IEEE/ACM International Conference on Automation of Software Test, AST, 2022, pp. 29–39, http://dx.doi.org/10.1145/3524481.3527233.

[83] Anshul Jindal, Vladimir Podolskiy, Michael Gerndt, Performance modeling for cloud microservice applications, in: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, ICPE '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 25–32, http://dx.doi.org/10.1145/3297663.3310309.

[84] Luca Giamattei, Antonio Guerriero, Roberto Pietrantuono, Stefano Russo, Assessing black-box test case generation techniques for microservices, in: Antonio Vallecillo, Joost Visser, Ricardo Pérez-Castillo (Eds.), Quality of Information and Communications Technology, Springer International Publishing, Cham, 2022, pp. 46–60.

[85] Luca Giamattei, Antonio Guerriero, Roberto Pietrantuono, Stefano Russo, Automated functional and robustness testing of microservice architectures, J. Syst. Softw. 207 (2024) 111857, http://dx.doi.org/10.1016/j.jss.2023.111857.

[86] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Wenhai Li, Chao Ji, Dan Ding, Delta debugging microservice systems, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 802–807, http://dx.doi.org/10.1145/3238147.3240730.

[87] Roberto Pietrantuono, Stefano Russo, Antonio Guerriero, Run-time reliability estimation of microservice architectures, in: 2018 IEEE 29th International Symposium on Software Reliability Engineering, ISSRE, 2018, pp. 25–35, http://dx.doi.org/10.1109/ISSRE.2018.00014.

[88] Roberto Pietrantuono, Stefano Russo, Antonio Guerriero, Testing microservice architectures for operational reliability, Softw. Test. Verif. Reliab. 30 (2) (2020) e1725, http://dx.doi.org/10.1002/stvr.1725, e1725 stvr.1725.

[89] Peter Alvaro, Kolton Andrus, Chris Sanden, Casey Rosenthal, Ali Basiri, Lorin Hochstein, Automating failure testing research at internet scale, in: Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 17–28, http://dx.doi.org/10.1145/2987550.2987555.

[90] Michael Assad, Christopher S. Meiklejohn, Heather Miller, Stephan Krusche, Can my microservice tolerate an unreliable database? Resilience testing with fault injection and visualization, in: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, in: ICSE-Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 54–58, http://dx.doi.org/10.1145/3639478.3640021.

[91] Hongyang Chen, Pengfei Chen, Guangba Yu, Xiaoyun Li, Zilong He, MicroFI: Non-intrusive and prioritized request-level fault injection for microservice applications, IEEE Trans. Dependable Secur. Comput. 21 (5) (2024) 4921–4938, http://dx.doi.org/10.1109/TDSC.2024.3363902.

[92] Christopher S. Meiklejohn, Andrea Estrada, Yiwen Song, Heather Miller, Rohan Padhye, Service-level fault injection testing, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 388–402, http://dx.doi.org/10.1145/3472883.3487005.

[93] Huayao Wu, Senyao Yu, Xintao Niu, Changhai Nie, Yu Pei, Qiang He, Yun Yang, Enhancing fault injection testing of service systems via fault-tolerance bottleneck, IEEE Trans. Softw. Eng. 49 (8) (2023) 4097–4114, http://dx.doi.org/10.1109/TSE.2023.3285357.

[94] Na Wu, Decheng Zuo, Zhan Zhang, An extensible fault tolerance testing framework for microservice-based cloud applications, in: Proceedings of the 4th International Conference on Communication and Information Processing, ICCIP '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 38–42, http://dx.doi.org/10.1145/3290420.3290476.

[95] Tianyi Yang, Cheryl Lee, Jiacheng Shen, Yuxin Su, Cong Feng, Yongqiang Yang, Michael R. Lyu, MicroRes: Versatile resilience profiling in microservices via degradation dissemination indexing, in: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, in: ISSTA 2024, Association for Computing Machinery, New York, NY, USA, 2024, pp. 325–337, http://dx.doi.org/10.1145/3650212.3652131.

[96] André De Camargo, Ivan Salvadori, Ronaldo dos Santos Mello, Frank Siqueira, An architecture to automate performance tests on microservices, in: Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services, 2016, pp. 422–429.

[97] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, Dan Pei, Localizing failure root causes

in a microservice through causality inference, in: 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), 2020, pp. 1–10, http://dx.doi.org/10.1109/IWQoS49365.2020.9213058.

[98] Peter Alvaro, Joshua Rosen, Joseph M. Hellerstein, Lineage-driven fault injection, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 331–346, http://dx.doi.org/10.1145/2723372.2723711.

[99] Orges Cico, Letizia Jaccheri, Anh Nguyen-Duc, He Zhang, Exploring the intersection between software industry and Software Engineering education-A systematic mapping of Software Engineering Trends, J. Syst. Softw. 172 (2021) 110736.

[100] Justus Bogner, Roberto Verdecchia, Ilias Gerostathopoulos, Characterizing technical debt and antipatterns in AI-based systems: A systematic mapping study, in: 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), IEEE, 2021, pp. 64–73.

[101] Roberto Verdecchia, Ivano Malavolta, Patricia Lago, Architectural technical debt identification: The research landscape, in: Proceedings of the 2018 International Conference on Technical Debt, 2018, pp. 11–20.

[102] Paolo Di Francesco, Ivano Malavolta, Patricia Lago, Research on architecting microservices: Trends, focus, and potential for industrial adoption, in: 2017 IEEE International Conference on Software Architecture, ICSA, IEEE, 2017, pp. 21–30.

[103] Antonia Bertolino, Software testing research: Achievements, challenges, dreams, in: Future of Software Engineering, FOSE'07, IEEE, 2007, pp. 85–103.

[104] Stefan Fischer, Pirmin Urbanke, Rudolf Ramler, Monika Steidl, Michael Felderer, An overview of microservice-based systems used for evaluation in testing and monitoring: A systematic mapping study, in: Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024), AST '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 182–192, http://dx.doi.org/10.1145/3644032.3644445.

[105] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén, et al., Experimentation in Software Engineering, vol. 236, Springer, 2012.

[106] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, Alexander Chatzigeorgiou, Identifying, categorizing and mitigating threats to validity in software engineering secondary studies, Inf. Softw. Technol. 106 (2019) 201–230, http://dx.doi.org/10.1016/j.infsof.2018.10.006, URL https://www.sciencedirect.com/science/article/pii/S0950584918302106.

[107] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, Stephen Linkman, Systematic literature reviews in software engineering – a systematic literature review, Inf. Softw. Technol. 51 (1) (2009) 7–15, http://dx.doi.org/10.1016/j.infsof.2008.09.009, URL https://www.sciencedirect.com/science/article/pii/S0950584908001390. Special Section - Most Cited Articles in 2002 and Regular Research Papers.

[108] Michael Gusenbauer, Neal R. Haddaway, Which academic search systems are suitable for systematic reviews or meta-analyses? Evaluating retrieval qualities of google scholar, PubMed, and 26 other resources, Res. Synth. Methods 11 (2) (2020) 181–217.

[109] Jan Piasecki, Marcin Waligora, Vilius Dranseika, Google search as an additional source in systematic reviews, Sci. Eng. Ethics 24 (2018) 809–810.

[110] Francisco Ponce, Jacopo Soldani, Hernán Astudillo, Antonio Brogi, Smells and refactorings for microservices security: A multivocal literature review, J. Syst. Softw. 192 (2022) 111393, http://dx.doi.org/10.1016/j.jss.2022.111393.