Journal of Software Engineering Research and Development, 2019, 6:1, doi: 10.5753/jserd.2019.xxx

From Detection to Refactoring of Microservice Bad Smells: A Systematic Literature Review

Mateus Dutra [**Federal University of Minas Gerais** | ***mateusmdutra@ufmg.br***]
Denis Pinheiro [**Federal University of Minas Gerais** | ***denis.pinheiro@dcc.ufmg.br.br***]
Johnatan Oliveira [**Federal University of Lavras** | ***johnatan.oliveira@ufla.br***]
Eduardo Figueiredo [**Federal University of Minas Gerais** | ***figueiredo@dcc.ufmg.br***]

Abstract

The extensive adoption of microservices architecture by technology companies is driven by its expected advantages, such as scalability, simplicity of development, and resilience, likely due to its cloud-native nature. However, the increasing complexity associated with this architecture can lead to the emergence of microservice smells, analogous to code smells, indicating potential architectural design issues. Despite the identification of numerous microservice smells in the literature, cohesive documentation to support architects detecting and refactoring them. This study conducts a Systematic Literature Review (SLR) to deepen the understanding of these microservice smells, explore detection tools and identify refactoring strategies to mitigate them. We conducted searches across six popular digital libraries, analyzing 27 relevant papers. As a result, we cataloged 104 distinct microservice bad smells, identified 7 detection tools, and compiled refactoring strategies for the most prevalent smells. This documentation aims to assist engineers and architects in identifying and effectively addressing microservice bad smells, thereby enhancing the quality and maintainability of microservice-based systems.

Keywords: *Microservice, Bad Smells, Refactoring, SLR*

1 Introduction

Microservice is an architectural style emerging out of service-oriented architecture, emphasizing self-management and reusable components as the means to improve software agility, scalability, fault tolerance, ease of deployment, maintainability, and autonomy (Jamshidi et al., 2018; De Lauretis, 2019; Alshuqayran et al., 2016). Microservices focus heavily on loose coupling and high cohesion of services, which are expected to improve the scalability, adaptability, and overall quality of software architectures (Rademacher et al., 2019). Given the extensive integration within major technology enterprises (Bogner, 2019; Taibi and Lenarduzzi, 2018), ensuring the quality of microservices becomes a priority.

Architectural smells are symptoms of poor design that can cause problems with code understandability and decrease maintainability (Taibi and Lenarduzzi, 2018; Garcia et al., 2009a). Several architectural smells have been defined in the literature for both generic and specific architectures, such as Shared Database (Garcia et al., 2009b; Taibi and Lenarduzzi, 2018; Taibi et al., 2018). However, cloud-native applications based on microservices can be affected by different types of issues (Taibi and Lenarduzzi, 2018), and the research field for service-based antipatterns and microservice smells is not as cohesive and organized (Bogner et al., 2019a).

To fill this gap, this paper extends our previous Systematic Literature Review (SLR) (Pinheiro et al., 2022) in two main ways. First, we updated the research to include all the new relevant papers on microservice bad smells and detection tools published between 2021 and 2024, which were not covered in the original work. Additionally, we compiled the *refactoring* techniques for the most common smells, since the previous SLR focused only on smell *detection*.

Although our previous work (Pinheiro et al., 2022) provided a valuable foundation by identifying key microservice bad smells and related detection tools, the field has evolved significantly in recent years. Several new tools, smell categories, and deployment contexts—such as Kubernetes-native environments—have emerged. Moreover, the 2021 review did not address refactoring strategies, which are essential for practical application and remain underexplored in the literature (Guo and Wu, 2021).

While the 2021 SLR (Guo and Wu, 2021) presented preliminary findings and a partial set of results, this study significantly updates and consolidates that work by expanding the number of reviewed studies, integrating newly proposed detection tools, and introducing systematic refactoring strategies for the most frequent microservice smells. Given the growing volume of literature and the need for updated, actionable guidance, this extension was necessary to provide a more comprehensive and current overview of the state of the art in microservice architecture quality.

To update and address this concerns, our research consolidates and expands the knowledge about microservice bad smells, supporting both their detection and resolution through systematic refactoring strategies. More precisely, this work has the following specific goals: (i) identify and catalog microservice bad smells reported in the literature, (ii) investigate and classify tools available for detecting these smells, and (iii) compile and associate refactoring techniques to address the most frequent microservice smells.

We found 27 primary studies related to microservice smells and detection tools. Additionally, we collected 104 different smells, 7 detection tools, and 19 related tools commonly used to automate the detection of architecture smells and to support the adoption of microservices. We also compiled refactoring guides for the 15 most frequent smells. The main contribution of this paper is to assist engineers and ar-

chitects in identifying microservice bad smells and, through refactoring strategies, provide guidance on how to address them effectively.

The paper is organized as follows. Section 2 presents background and related work about microservice bad smells. Section 3 describes the methods used in the research. Section 4 shows and discusses the results of the SLR in the light of three research requestions. Section 5 lists possibles threats to validity. Finally, Section 6 concludes the manuscript and provides guidance for future works.

2 Related Work

Several research papers have investigated bad smells in different software development artifacts, such as source code (Santana et al., 2024) and design models (Cardoso and Figueiredo, 2015). This section focuses on related work on microservices bad smells, organized by relevance, proximity to the topics discussed in each study, and their category (Empirical Studies, Systematic Reviews, and Tool Proposal).

2.1 Empirical Studies

Arcelli Fontana et al. (2023) explore the impact of architectural smells on software performance, focusing on how the removal of two common architectural smells, God Class and Cyclic Dependency, affects performance metrics such as CPU usage and memory consumption. Their study uses tools like Arcan and Designite to detect these smells and runs experiments on two systems: OpenMRS (a monolithic system) and TeaStore (a microservice system). The results show improvements in performance after smell refactoring, with execution time reduced by up to 47% and memory consumption by up to 20%, indicating that addressing architectural smells can improve software performance. While Arcelli Fontana et al. (2023) focus on performance, particularly execution time and memory usage, our study takes a broader view by cataloging and analyzing 104 microservice-specific bad smells. Our review goes beyond performance impacts, delving into the tools and strategies for detecting and refactoring a wider variety of smells, covering 12 different categories.

Liu et al. (2021) conducted an empirical study on the relationship between runtime performance deficiencies and architectural bad smells (ABS) in microservice systems. They developed a Microservice Runtime Stress Testing Framework (MRSTF) to test three architectural bad smells—Dependency Circle, Poor Use of Abstract, and Shared Database—on the Train Ticket system. The study shows that these smells can negatively impact performance metrics like CPU and memory usage, as well as response times. Liu et al. focus on a narrow set of smells and their impact on performance, while our research catalogs 104 microservice-specific bad smells and addresses both detection and some refactoring strategies across a wider range of issues, not limited to performance.

2.2 Systematic Literature Reviews

Cerny et al. (2023) conducted a study that classifies recurring bad design practices, including antipatterns and bad smells, in microservice architectures. They identified 58 unique microservice antipatterns and grouped them into five categories: granularity, service interface, cohesion, inter-service decomposition, and service interaction. The study also discusses various methods for detecting these antipatterns through static and dynamic analysis. Cerny et al. (2023) provide a broad overview and classification of antipatterns, but they do not deeply explore refactoring strategies. In contrast, our work identifies a more extensive set of 104 microservice bad smells and focuses significantly on practical refactoring strategies to address these issues.

Guo and Wu (2021) conducted a systematic literature review focused on identifying bad smells in cloud-based applications and microservices. Their study proposed a classification of smells across different layers of cloud environments and presented a taxonomy covering application, infrastructure, and deployment smells. While their review provides a broader perspective on cloud computing systems, our work focuses specifically on microservice architecture and goes further by identifying refactoring strategies and detection tools. These differences highlight the contribution of our review in providing actionable guidance for microservice-specific architectural issues.

Table 1 presents a comparative analysis of four SLRs on bad microservices smells, namely Cerny et al. (2023), Guo and Wu (2021), Pinheiro et al. (2022), and the present work. The comparison encompasses the number of smells reported, the detection tools employed, and the refactoring strategies proposed. By situating our study with respect to prior research, Table 1 highlights the novel contributions of this work. For instance, while we documented 104 microservices smells, Pinheiro et al. (2022) found 47 ones. Furthermore, as far as we are concerned, we are the first SLR to study refactoring strategies for microservices smells.

2.3 Tool Proposals

Walker et al. (2020) present a method for detecting code smells in microservice architectures through static analysis. They introduce MSANose, a tool that detects 11 microservice-specific code smells, including Cyclic Dependency, Shared Persistency, and Microservice Greedy. The authors test this tool on two open-source systems, Train Ticket and Teacher Management System, demonstrating that static analysis can effectively identify architectural and design issues in distributed systems. Walker et al. (2020) contribution is focused on the development of a specific tool for automated code smell detection. In contrast, our study takes a broader perspective, offering a comprehensive overview of microservice bad smells, not just focusing on detection.

Pigazzini et al. (2020) propose an extension of a tool for detecting architectural smells, focusing on three microservice-specific smells: Cyclic Dependencies, Hard-Coded Endpoints, and Shared Persistence. They applied the extended tool, Arcan, to five open-source projects to validate

Table 1. Related Systematic Reviews Comparison

Paper	Title	Smells	Detection Tools	Refactoring Strategies
Cerny et al. (2023)	Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study	58	0	0
Guo and Wu (2021)	A review of bad smells in cloud-based applications and microservices	34	0	0
Pinheiro et al. (2022)	Microservice smells and automated detection tools: A systematic literature review	47	5	0
Our SLR	From Detection to Refactoring of Microservice Bad Smells: A Systematic Literature Review	104	7	15

the detection strategies. While their study effectively detects these smells through static analysis, it is limited to just three smells and does not address refactoring. In comparison, our study provides a broader framework, identifying 104 distinct microservice bad smells and placing significant emphasis on refactoring strategies, offering more practical solutions for handling a wider range of microservice smells.

In addressing security smells, Ponce et al. (2022) propose a method called TriSS (Triage Security Smells), which helps prioritize the resolution of microservice security smells by assigning urgency codes based on the business relevance of the services and the impact of the smells on security and other quality attributes, such as performance and maintainability. They focus on smells like centralized authorization, hard-coded secrets, and insufficient access control. TriSS helps practitioners systematically determine which security smells to address first, providing a method for triaging based on urgency. While this approach is essential for managing security-related smells, our work takes a broader approach by cataloging and examining 104 microservice-specific bad smells, including those not directly tied to security concerns. Our systematic review also emphasizes detection and refactoring strategies, providing more practical guidance on resolving these issues beyond just prioritizing them.

3 Methods

We conducted an SLR following well-known protocols and guidelines (Kitchenham and Charters, 2007). Furthermore, we considered updated practices (Kitchenham et al., 2015; Pfleeger and Kitchenham, 2025) to improve reporting quality and ensure alignment with the most recent standards for systematic literature reviews in software engineering. SLR is a method used to identify, evaluate, and interpret all available research on a specific topic (Kitchenham and Charters, 2007). This process, following well-defined guidelines (Kitchenham and Charters, 2007; Kitchenham et al., 2015; Pfleeger and Kitchenham, 2025), comprises two main stages: planning and execution. The primary goal of this SLR is to recognize and analyze documented microservices smells along with their corresponding detection tools and refactoring techniques.

This work expands the previous review (Pinheiro et al., 2022) by updating the analysis to include studies published between 2021 and 2024. Furthermore, we also expand the scope not only to catalog additional smells and detection tools but also to identify refactoring strategies for the most frequent smells. The updated timeframe ensures that the review remains comprehensive and reflects the latest developments in the field of microservices architecture quality.

3.1 Planning

Following the guidelines proposed by Kitchenham (Kitchenham and Charters, 2007), we planned the following steps: (1) identify the topics to be investigated (research questions); (2) select the digital libraries; (3) define the search string to retrieve relevant studies; and (4) apply a filtering process with inclusion/exclusion criteria. This study aims to address three main research questions related to the field of microservice bad smells presented below.

RQ1. How has the research of microservice smells evolved over time?

RQ2. What are the microservice smells detection tools presented in the literature?

RQ3. What are the refactoring techniques for microservice smells described by the literature?

RQ1 aims to improve our understanding of the current research landscape on microservice bad smells and how it has evolved over time. By answering RQ1, we expect not only to evaluate the maturity of this field but also to identify trends. For instance, results may indicate whether the community's attention to microservice smells is growing, stable, or declining. We may also observe emerging topics. This RQ was inspired by previous studies (Lenarduzzi et al., 2020).

With RQ2, we aim at identifying the detection tools available in the literature and their practical adoption (Fernandes et al., 2016). Engineers and architects rely on automated tools to identify potential issues in complex microservice-based systems. Therefore, our work can provide a comprehensive overview of the tools and their detection approaches. RQ2 is motivated by the lack of consolidated surveys listing and comparing detection tools for microservice smells.

Our RQ3 aims to compile refactoring techniques for microservice smells since detecting smells without proper guidance for their correction leaves a gap in the practical applicability. By answering this RQ, we aim to assist practitioners in effectively addressing architectural problems and improve their architectures. RQ3 stems from a limitation of our previ-

ous SLR (Pinheiro et al., 2022), which focused on detection but did not explore refactoring strategies.

Table 2 presents the list of digital libraries used to perform this SLR. The filtering process allows classifying each study under review as a candidate to be included or excluded from the SLR based on specific criteria. Table 3 presents the inclusion and exclusion criteria used to select the primary studies.

Table 2. Selected Digital Libraries

Database	Address
ACM Digital Library	`https://dl.acm.org`
IEEE Explore	`https://ieeexplore.ieee.org`
Engineering Village	`https://www.engineeringvillage.com`
Science Direct	`https://www.sciencedirect.com`
Scopus	`https://scopus.com`
Springer	`https://link.springer.com`

The definition of the search string was based on the research questions and aligned with guidelines from the literature (Kitchenham and Charters, 2007). In particular, we replicate the search string from our previous SLR (Pinheiro et al., 2022), which already established a validated set of keywords. That is, we reused their original search and empirically evaluated its outputs as detailed below.

$$microservice \wedge (architectural\ smell \vee code\ smell \vee bad\ smell \vee anti-pattern \vee technical\ debt)$$

We reused the search string of our previous SLR (Pinheiro et al., 2022) since it was aligned with our research scope and was effective in identifying relevant studies. To assess the accuracy and completeness of the search string, we empirically evaluated its results using the test set method. That is, we checked whether a known set of relevant papers (Taibi and Lenarduzzi, 2018; Bogner et al., 2018; Soldani. et al., 2023) appeared in the search results. Additionally, two researchers independently reviewed the initial search results to verify their relevance.

The execution of the search varied slightly depending on the database. In IEEE Xplore and Scopus, we used the advanced search interface to search in title, abstract, and keywords. In databases that did not offer this granularity, such as ACM Digital Library, we searched in the entire metadata. All searches were restricted to papers written in English and from the Computer Science field. We applied filters manually when available (e.g., the language and subject area filters available in Scopus).

3.2 Execution

Figure 1 illustrates the data extraction process following the PRISMA flow diagram (Page et al., 2021). This process was conducted using the selected digital libraries listed in Table 2. The search results were independently examined and screened by two authors.

The execution of these search steps was guided to fulfill and respond to the research questions. Step 1 involved identifying all papers containing the keywords *microservice* and *architecture smells* (or their synonyms) somewhere in the paper text, including all metadata. This search, considering all published works, resulted in 1,887 papers. In Step 2, we filtered out all duplicate papers and applied the inclusion and exclusion criteria, resulting in 1,331 papers. In Step 3, we analyzed the titles of the remaining papers, which reduced the count to 529. Step 4 involved analyzing the abstracts of these papers, further narrowing the count to 162. Finally, in Step 5, we conducted a full-text review, selecting 27 primary studies that reported on microservice smells, their detection tools, or refactoring techniques.

Table 3. Filter criteria

Inclusion Criteria	Exclusion Criteria
Papers published in Computer Science	Papers shorter than five pages
Papers written in English	Thesis, dissertations, tutorials and grey literature
Papers available in electronic format	
Propose or cite microservice bad smell	

4 Results and Discussion

This section presents the findings of our systematic literature review on microservice bad smells. First, we provide an overview of the primary studies selected. Subsequently, we delve into the evolution of research on microservice smells, the detection tools identified in the literature, and the refactoring techniques proposed to mitigate these smells. Finally, we discuss the application of the detection and refactoring for practitioners, such as developers and architects. Each subsection aims to provide a comprehensive understanding of these aspects, contributing to a holistic view of the current state of research in this domain.

On our supplementary website [1], we have compiled and organized all 104 microservice smells identified across the 27 studies. These smells are grouped into 12 distinct categories to simplify their identification: API Management, Dependency Management, Middleware, Discovery, Data Management, Decomposition, Team/Product Management, Architectural Standards, Quality Assurance, DevOps (CI/CD), Documentation, and Migration. This categorization aims to offer a clearer structure and better understanding of the different types of smells.

4.1 Trends in Microservice Bad Smells Research

This section discusses the results for the first research question.

RQ1- How has the research of microservice smells evolved over time?

[1]The exhaustive list of 104 microservice smells is available at `https://github.com/mateusmdutra/microservice-bad-smells/blob/main/complete-list-smells.pdf`.

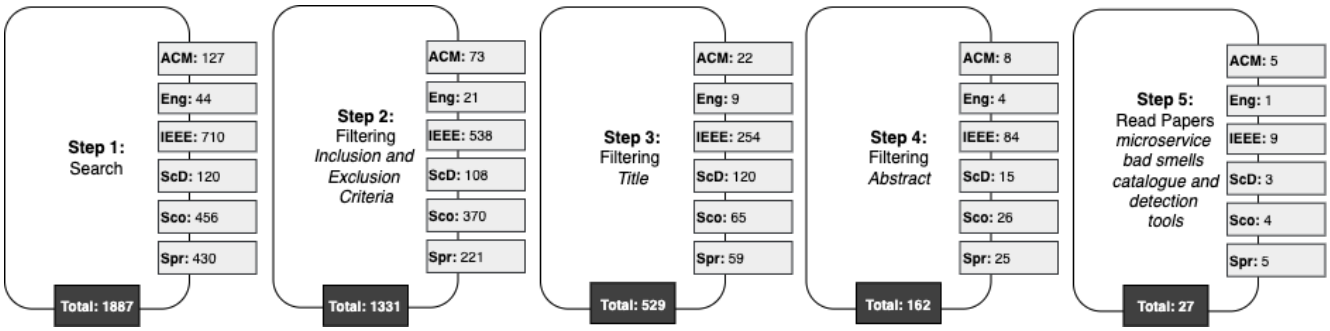

Figure 1. Research execution

Figure 2 (A) shows the overall raising interest given the publication years of the selected primary studies that presented research related to microservice smells. The research interest in this area began in 2018, with three notable papers published that year (Taibi and Lenarduzzi, 2018; Carrasco et al., 2018; Bogner et al., 2018). Since then, the field has seen consistent interest, peaking in 2020 with seven papers. Although Figure 2 shows a stable level of interest over recent years, the topic remains underrepresented in the literature. This underscores the needs for further in-depth studies and innovative approaches to advance the understanding of microservice smells.

Analyzing the publication venues and the number of publications in each, we found that TechDebt, JSS, and LNBIP had the most publications, with a total of three each. Following these, ICSA, SEAA, and EASE had two publications each among the selected primary studies. All other publication venues had one publication each, including recognized software engineering conferences, such as ICSME and various scientific journals.

Figure 2 (B) presents the research types identified in the selected studies. We considered following research types: *Survey* as the research performed by using a questionnaire or interviewing participants (Viggiato et al., 2018); *New Approach*, research proposing a new method that solves a common problem in microservice-based systems; *Case study*, research that studied a real or hypothetical problem related to microservice smells, *Tool presentation*, research presenting a new tool or a new version of a tool; and *Review*, research that made a scientific review of white and gray literature.

We found 4 papers that published the result of *Surveys* and 6 studies that presented *New Approaches*. For instance, Taibi and Lenarduzzi (2018) presented at first time the 11 microservice smells catalog as a result of interviewing 72 developers with experience in developing microservice-based systems. As another example, Gaidels and Kirikova (2020) presented a service dependency graph analysis in the process of identifying microservice smells.

We found 7 works that presented *Case study* as research method. For instance, Toledo et al. (2020) performed a multiple case study in 4 international companies, supported by 6 architects and, as a result, the research reported 9 issues related to Shared Library and 2 solutions were proposed. Eight (8) works performed a *Tool presentation* of automated microservice smells detection. Other two (2) researches presented a *Review* of white and gray literature related to architecture design and the migration process of microservice-based systems.

Notably, we can observe that the initial 11 microservice smells, proposed by Taibi and Lenarduzzi (2018), are prevalent across the majority of the reviewed literature (Table 6), with the detection tools covering most of these identified smells (Section 4.2). This finding emphasizes the significance and persistence of these issues in microservice architecture. The most frequently cited smells in the papers were Not Having an API Gateway (Ntentos et al., 2021; Tighilt et al., 2023; Soldani. et al., 2023; Soldani et al., 2021; Taibi and Lenarduzzi, 2018), Cyclic Dependency (Taibi and Lenarduzzi, 2018; Pigazzini et al., 2022; Tighilt et al., 2023; Gamage and Perera, 2021) and Shared Database (Taibi and Lenarduzzi, 2018; Walker et al., 2021; Tighilt et al., 2023; Soldani et al., 2021).

Main Findings – RQ1: Smell Landscape

We summarized the landscape of publications about microservices bad smell by year and types of researches. We observed that the number of publications has increased from 2018 to 2023. Furthermore, the most common type of publication is work presenting new tools.

4.2 Overview of Detection Tools

This section discusses the results for the second research question.

RQ2- What are the microservice smells detection tools presented in the literature?

There are several approaches for detecting microservice smells. The first and most intuitive method is static analysis of the code and configuration files of microservices. This approach identifies potential smells, such as Endpoint-based Communication, Nano Services, and Shared Libraries. Another method is dynamic detection, which involves analyzing distributed tracing in running applications or collecting application metrics.

For static analysis, Pigazzini et al. (2020) introduced a new version of ***Arcan***, which detects three microservices smells from the Taibi and Lenarduzzi catalogue (Taibi and Lenarduzzi, 2018). Walker et al. (2021) presented the ***MSANose*** tool, capable of detecting 11 microservice smells reported by Taibi and Lenarduzzi (Taibi and Lenarduzzi, 2018). MSANose was evaluated for detection accuracy using two benchmark projects (Walker et al., 2021). Addition-

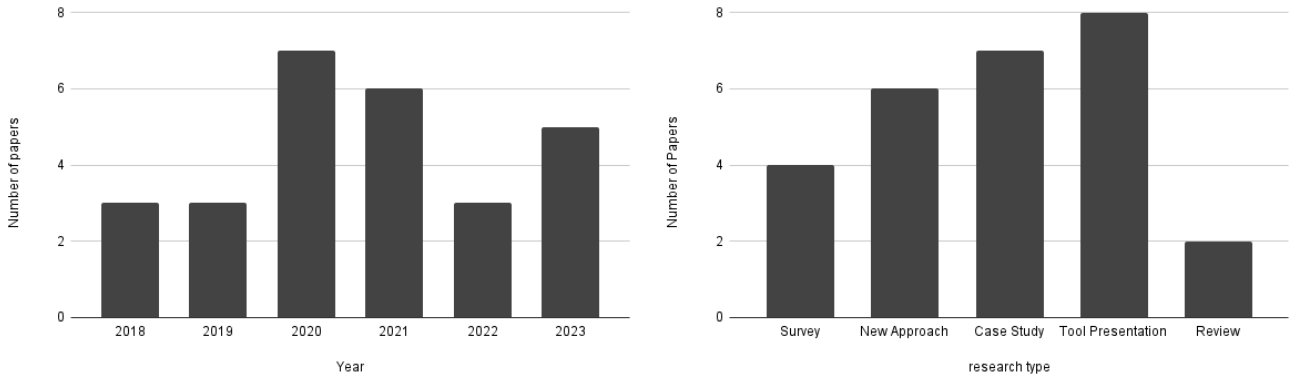

Figure 2. (A) Publications by Year and (B) Research Types

ally, Capilla et al. (2023) and Fang et al. (2023) utilized the ***Designite tool*** [2] to detect various microservice smells. Similarly, Tighilt et al. (2023) proposed ***MARS*** (Microservice Antipatterns Research Software), a tool that detects 16 microservice smells by analyzing source code, environment files, configuration files, deployment files, databases, docker images, HTTP requests, and imports. In Kubernetes environments, Soldani et al. (2021); Soldani. et al. (2023) introduced two tools: ***µTOSCA toolchain***, which analyzes, detects, and resolves microservice smells in existing Kubernetes environments, and ***KubeFreshener***, which detects and helps refactor microservice smells in Kubernetes deployment units.

Regarding dynamic analysis, Pigazzini et al. (2022) extended ***Arcan*** to support dynamic analysis and compared the results with the existing static analysis tool. Gamage and Perera (2021) proposed the ***MAIG*** tool for detecting 5 microservice bad smells through runtime data, automatically generating up-to-date dependency models with metrics. Moreover, microservice-based systems, like other software systems with source code, can introduce both architectural and code smells. These may be identified by regular detection tools such as *SonarQube* [3]. Bogner et al. (2018, 2019b) conducted two studies on 19 tools that support and address symptoms of low maintainability and the challenges of evolving microservices.

Main Findings – RQ2: Detection Tools

We identified 7 specific tools for detecting microservice bad smells. Most tools rely on static analysis, but a few tools also support dynamic analysis. MSANose and Arcan are the most cited tools, but the newest tools focus on cloud-native contexts (e.g., Kubernetes).

4.3 Refactoring Techniques

This section discusses the results for our third research question.

RQ3- What are the refactoring techniques for microservice smells described by the literature?

As noted in Section 3.2, some of the 104 identified smells show interrelations. Table 6 presents a reduced list of 15 microservice smells found in the selected studies. In this paper, we focus on smells that: (i) were reported by two or more studies and (ii) were identified by a detection tool. The first column contains the microservice bad smell name; the second column lists the detection tools available for each smell; and the third column describes refactoring to guide architects in mitigating them inside their microservices systems.

Table 6, shows three of the most related smells (Cyclic Dependency, No API Gateway and Shared Database) are covered by the majority of the detection tools. The Arcan tool detects all selected smells. Notably, with the extension on Arcan, made by Pigazzini et al. (2022), two dependencies-related smells (Cyclic Dependency and Hub-like Dependency) can be detected using both static and dynamic analysis.

The following studies discuss refactoring strategies for microservice bad smells: Taibi and Lenarduzzi (2018), Soldani. et al. (2023), Soldani et al. (2021) and Pulnil and Senivongse (2022). They examine refactorings for various smells, including Cyclic Dependency, Microservice Greedy and Wrong Cuts, highlighting important microservice architecture principles such as modularization, separation of concerns, and responsibility assignment. Furthermore, Endpoint-Based Service Interaction and Hard-Coded Endpoints represent two related smells that necessitate distinct refactoring approaches. The former focuses on minimizing the number of endpoint calls, introducing event-driven architecture or message queues, and the latter seeks to eliminate explicit endpoint declarations by promoting the use of configuration-based endpoint discovery.

Hub-like Dependency (Capilla et al., 2023; Arcelli Fontana et al., 2023; Pigazzini et al., 2022) was a well-discussed microservice bad smell. However, the selected studies do not provide an explicit refactoring guide for addressing it. Given the definition of this smells — where classes or modules (representing other services in a microservice architecture) has excessive responsibilities and depend on numerous other components — we propose a corresponding refactoring strategy based on ESB Usage. This strategy similarly aims to decentralize communication among services.

Nano Service was discussed in two studies (Tighilt et al.,

[2] https://designite-tools.com/
[3] https://www.sonarsource.com/products/sonarqube/

Table 4. Detection tools for microservice bad smells: scope and coverage

Tool	Purpose	Type of Analysis	Functional Scope and Smells Detected	Reference
Arcan	Detect architectural smells and violations	Static / Dynamic	Analyzes service dependencies and modularity; detects 3 smells (e.g., Cyclic Dependency, Shared Persistence); extended to dynamic analysis in 2022	Pigazzini et al. (2020, 2022)
MSANose	Identify microservice-specific smells	Static	Applies static code analysis in microservices; detects 11 smells from Taibi and Lenarduzzi (2018), including Microservice Greedy and Shared Persistency	Walker et al. (2021)
Designite	Identify design and architectural smells	Static	Detects various design smells (e.g., God Class, Cyclic Dependency); not exclusive to microservices	Capilla et al. (2023)
MARS	Detect microservice antipatterns and smells	Static	Analyzes code, config files, docker images, and HTTP requests; detects 16 smells using rule-based analysis	Tighilt et al. (2023)
µTOSCA	Analyze and refactor microservice deployments	Static	Focuses on Kubernetes environments; identifies and resolves deployment-related smells; smell count not specified	Soldani et al. (2021)
KubeFreshener	Refactor Kubernetes deployments	Static	Detects and suggests refactorings for smells in Kubernetes YAML deployment files	Soldani. et al. (2023)
MAIG	Detect runtime smell patterns	Dynamic	Builds runtime dependency graphs; detects 5 dynamic architectural smells related to service coupling	Gamage and Perera (2021)
SonarQube	General code analysis	Static	Detects generic code smells, bugs, and vulnerabilities; not specific to microservices	Bogner et al. (2018, 2019b)

2023; Gamage and Perera, 2021). Similar to the Hub-like Dependency smell, Nano Service did not have a refactoring proposal in the collected papers. For this smell, we propose a refactoring approach analogous to Fowler's 'Inline Method' for code smells (Fowler, 1999). In this approach, services that perform minimal functions are recommended to be merged into a larger service with well-defined responsibilities. Microservice Greedy has the same refactoring as Nano Service (Taibi and Lenarduzzi, 2018).

Table 5 presents a selection of the most frequently discussed microservice smells along with their respective detection tools and proposed refactoring strategies.

The strategies discussed above serve not only as academic insights but also as practical guidance for software professionals. The following section explores how these findings can be applied in real-world development environments.

Main Findings – RQ3: Refactoring Strategies

We compiled refactoring strategies for the 15 most cited microservice smells. Some smells, such as Cyclic Dependency and Shared Database, are supported by most detection tools. However, other smells (e.g., Nano Service) lacked refactoring techniques in the literature — we proposed them based on code-level analogies. Most strategies emphasize modularization, service boundaries, and configuration improvements.

4.4 Implications for Practitioners and Researchers

The refactoring list and related detection tools are a practical reference for microservices practitioners. Both architects and developers can use the proposed refactorings to improve existing microservices applications, or use them as guidelines on architecting and developing new microservices applications. The detection tools can be explored and configured on build pipelines as an automatic detector of microservices bad smells, in a fast failure workflow, by improving internal quality of microservices application during quality assurance phase of a DevOps pipeline. To support adoption by industry professionals, we have made the complete catalog of 104 microservice bad smells publicly available on GitHub[4].

In addition, the structured dataset presented in this study also serves as a solid foundation for researchers interested in empirical validation, tool evaluation, or extending the taxonomy of microservice architectural problems. This resource can also serve as a practical reference during code reviews, architectural assessments, or as part of DevOps pipelines. Tools discussed in Section 4.2 may be incorporated into CI/CD workflows to automate the detection of architectural issues. We encourage researchers, developers, and architects to adapt and extend these resources according to their particular interests (e.g., research and project contexts).

[4] https://github.com/mateusmdutra/microservice-bad-smells/blob/main/complete-list-smells.pdf

Table 5. Selected microservice bad smells and proposed refactorings

Smell	Tools Available	Refactoring Strategy
No API Gateway	µTOSCA, KubeFreshener	Introduce a centralized API gateway to handle routing, authentication, rate limiting
Hard-Coded Endpoints	Arcan, MSANose	Replace static endpoint calls with configuration-based dynamic discovery
Nano Service	Arcan, MAIG	Merge very small services into larger cohesive units ("Inline Method" approach)
Shared Database	MSANose, MARS	Decouple services by migrating to separate databases per service
Microservice Greedy	Arcan, MSANose	Split services with excessive responsibilities into well-scoped components

5 Threats to Validity

The main threat when performing the SLR is the validity of the results. We discussed the SLR validity with respect to the 4 groups of common threats to validity: internal, construct, external, and conclusion (Wohlin, 2000).

Internal validity: A potential limitation of this study is that if it were replicated by another team of researchers, some studies excluded in this review might be included, and others currently included could be excluded. This study has been addressed by involving four researchers, and by a protocol that was piloted and evaluated. However, in general, we believe that the internal validity of the SLR is high given the use of a systematic procedure, consultation with the researchers in the field, involvement, and discussion between the four researchers.

Construct validity: The four authors of this study are researchers in the software engineering field. We are not aware of any bias we may have introduced during the analyses. However, from the reviewer's perspective, a construct validity threat could be biased judgment. In this study, the decision of which studies to include or to exclude and how to categorize the studies could be biased and thus pose a threat. A possible threat in such a review is to exclude some relevant papers. To minimize this threat both the processes of inclusion and exclusion were piloted by three coauthors. Furthermore, potentially relevant studies that were excluded were documented. This way, we believe that we do not have omitted relevant papers.

External validity: The identified primary studies may not generalize. The search for the papers was conducted in 6 relevant scientific databases in order to capture as much as possible the available papers. However, the quality of search engines could have influenced the completeness of the identified primary studies. That means our search may have missed those studies whose authors would have used other terms to specify the microservice bad smells, or may not have used the keywords that we used for searches in the title, abstract, and keywords of their papers.

Conclusion validity: Furthermore, from the authors' perspective, a potential threat to *conclusion validity* is the reliability of the data extraction from the primary studies, since not all information was obvious to answer the research questions and some data had to be inferred. Therefore, in order to ensure conclusion validity, sometimes cross-discussions among the paper authors took place to reach a common agreement. Furthermore, in the event of a disagreement between the two researchers, a third author acted as an arbitrator to ensure a position to be reached.

6 Conclusion and Future Work

This paper presents a systematic literature review on microservice bad smells and detection tools. We found that interest in microservice bad smells has been increasing since 2018, with surveys being the most common research type published in recognized scientific venues, such as TecDebt, ICSA, and IEEE Software. We identified 104 microservice bad smells categorized by the primary studies, all representing symptoms of design or implementation flaws in microservice projects. Additionally, we found seven detection tools that identify the initial smells presented by Taibi and Lenarduzzi (2018), as well as some discovered later. Finally, to mitigate the most common smells, we created a list of recommended refactoring strategies along with the available detection tools for each smell.

As future work, we intend to compile a guideline based on the reported microservice smells to help architects identify and avoid bad practices during the development of microservices. Furthermore, we aim to deepen the dynamic analysis of microservice bad smells, combining metrics collection with quality analysis. Moreover, we also plan to carry out a case study comparing the detection results of the different tools found, in order to see the validity of each one of them for each case. Additionally, we plan to conduct a survey with practitioners to gather insights on how detection tools and the refactoring strategies can be effectively incorporated into their daily workflows.

Acknowledgements

This research was partially supported by Brazilian funding agencies: CNPq (Grant 305171/2025-9), CAPES, and FAPEMIG (Grant APQ-01488-24).

References

Alshuqayran, N., Ali, N., and Evans, R. (2016). A systematic mapping study in microservice architecture. In *2016*

IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), pages 44–51.

Arcelli Fontana, F., Camilli, M., Rendina, D., Taraboi, A. G., and Trubiani, C. (2023). Impact of architectural smells on software performance: an exploratory study. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, pages 22–31.

Bogner, J. (2019). Microservices in industry: insights into technologies, characteristics, and software quality. *IEEE international conference on software architecture companion (ICSA-C)*.

Bogner, J., Boceck, T., Popp, M., Tschechlov, D., Wagner, S., and Zimmermann, A. (2019a). Towards a collaborative repository for the documentation of service-based antipatterns and bad smells. In *International Conference on Software Architecture Companion (ICSA-C)*, pages 95–101.

Bogner, J., Fritzsch, J., Wagner, S., and Zimmermann, A. (2018). Limiting technical debt with maintainability assurance: An industry survey on used techniques and differences with service- and microservice-based systems. In *International Conference on Technical Debt*, page 125–133.

Bogner, J., Fritzsch, J., Wagner, S., and Zimmermann, A. (2019b). Assuring the evolvability of microservices: Insights into industry practices and challenges. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 546–556.

Capilla, R., Fontana, F. A., Mikkonen, T., Bacchiega, P., and Salamanca, V. (2023). Detecting architecture debt in micro-service open-source projects. In *49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, page 394 – 401.

Cardoso, B. and Figueiredo, E. (2015). Co-occurrence of design patterns and bad smells in software systems: An exploratory study. In *Proceedings of the Brazilian Symposium on Information Systems (SBSI)*.

Carrasco, A., Van Bladel, B., and Demeyer, S. (2018). Migrating towards microservices: Migration and architecture smells. In *International Workshop on Refactoring (IWoR/ASE)*, pages 1 – 6.

Cerny, T., Abdelfattah, A. S., Maruf, A. A., Janes, A., and Taibi, D. (2023). Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study. *Journal of Systems and Software*, 206.

De Lauretis, L. (2019). From monolithic architecture to microservices architecture. pages 93–96.

Fang, H., Cai, Y., Kazman, R., and Lefever, J. (2023). Identifying anti-patterns in distributed systems with heterogeneous dependencies. In *IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., and Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. In *20th International Conference on Evaluation and Assessment in Software Engineering (EASE)*.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Boston, MA.

Gaidels, E. and Kirikova, M. (2020). Service dependency graph analysis in microservice architecture. In *Proceedings of the 19th International Conference on Business Informatics Research (BIR)*, pages 128–139. Springer.

Gamage, I. U. P. and Perera, I. (2021). Using dependency graph and graph theory concepts to identify anti-patterns in a microservices system: A tool-based approach. In *Moratuwa Engineering Research Conference (MERCon)*.

Garcia, J., Popescu, D., Edwards, G., and Medvidovic, N. (2009a). Identifying architectural bad smells. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 255–258.

Garcia, J., Popescu, D., Edwards, G., and Medvidovic, N. (2009b). *Toward a Catalogue of Architectural Bad Smells*, pages 146–162. Springer.

Guo, D. and Wu, H. (2021). A review of bad smells in cloud-based applications and microservices. In *2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*, pages 255–259.

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., and Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35.

Kitchenham, B. A., Budgen, D., and Brereton, P. (2015). *Evidence-Based Software Engineering and Systematic Reviews*. Chapman & Hall/CRC.

Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001.

Lenarduzzi, V., Lomio, F., Saarimäki, N., and Taibi, D. (2020). Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*, 169:110710.

Liu, L., Tu, Z., He, X., Xu, X., and Wang, Z. (2021). 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)*, pages 751–757.

Neri, D., Soldani, J., Zimmermann, O., and Brogi, A. (2020). Design principles, architectural smells and refactorings for microservices: a multivocal review. volume 35.

Ntentos, E., Zdun, U., Plakidas, K., and Geiger, S. (2021). Evaluating and improving microservice architecture conformance to architectural design decisions. In *Service-Oriented Computing*, page 188–203.

Page, M. J. et al. (2021). The prisma 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372.

Pfleeger, S. L. and Kitchenham, B. (2025). Evidence-based software engineering guidelines revisited. *IEEE Transactions on Software Engineering*.

Pigazzini, I., Di Nucci, D., Fontana, F. A., and Belotti, M. (2022). Exploiting dynamic analysis for architectural smell detection: a preliminary study. In *48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, page 282 – 289.

Pigazzini, I., Fontana, F. A., Lenarduzzi, V., and Taibi, D. (2020). Towards microservice smells detection. In *Proceedings of the 3rd International Conference on Technical Debt*, pages 92–97.

Pinheiro, D., Oliveira, J., and Figueiredo, E. (2022). Microservice smells and automated detection tools: A systematic literature review. In *In proceedings of the Inter-*

national Conference on Microservices (Microservices).

Ponce, F., Soldani, J., Astudillo, H., and Brogi, A. (2022). Smells and refactorings for microservices security: A multivocal literature review. *Journal of Systems and Software*, 192:111393.

Pulnil, S. and Senivongse, T. (2022). A microservices quality model based on microservices anti-patterns. In *19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*.

Rademacher, F., Sachweh, S., and Zündorf, A. (2019). Aspect-oriented modeling of technology heterogeneity in microservice architecture. In *International Conference on Software Architecture (ICSA)*.

Santana, A., Figueiredo, E., Pereira, J. A., and Garcia, A. (2024). An exploratory evaluation of code smell agglomerations. *Software Quality Journal*.

Soldani., J., Marinò., M., and Brogi., A. (2023). Semi-automated smell resolution in kubernetes-deployed microservices. In *Proceedings of the 13th International Conference on Cloud Computing and Services Science - CLOSER*, pages 34–45. INSTICC, SciTePress.

Soldani, J., Muntoni, G., Neri, D., and Brogi, A. (2021). The µtosca toolchain: Mining, analyzing, and refactoring microservice-based architectures. *Software - Practice and Experience*, 51(7):1591 – 1621.

Taibi, D. and Lenarduzzi, V. (2018). On the definition of microservice bad smells. *IEEE Software*.

Taibi, D., Lenarduzzi, V., and Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. In *Int'l Conf. on Cloud Computing and Services Science (CLOSER)*.

Tighilt, R. et al. (2023). On the maintenance support for microservice-based systems through the specification and the detection of microservice antipatterns. *Journal of Systems and Software*, 204:111755.

Toledo, S., Martini, A., and Sjøberg, D. (2020). Improving agility by managing shared libraries in microservices. *Lecture Notes in Business Information Processing (LNBIP)*.

Viggiato, M., Terra, R., Rocha, H., Valente, M. T., and Figueiredo, E. (2018). Microservices in practice: A survey study. In *6th Workshop on Software Visualization, Evolution and Maintenance (VEM)*.

Walker, A., Das, D., and Cerny, T. (2020). Automated code-smell detection in microservices through static analysis: A case study. *Applied Sciences*, 10(21):7800.

Walker, A., Das, D., and Cerny, T. (2021). Automated microservice code-smell detection. In *Information Science and Applications*, page 211–221. Springer Singapore.

Wohlin, C. (2000). *Experimentation in Software Engineering: An Introduction*. The sKluwer ISECS.

A Appendix

Table 6. Microservice Bad Smells refactorings

Smell	Tools	Refactoring	Studies
API Versioning	Arcan, MARS, MSANose	Update the API design to support multiple versions, allowing backward compatibility and smoother transitions for clients	(Taibi and Lenarduzzi, 2018; Pulnil and Senivongse, 2022)
Cyclic Dependency	Arcan, Designite, KubeFreshener, MAIG, MARS, MSANose	Restructure services to remove circular dependencies via modularization	(Taibi and Lenarduzzi, 2018; Soldani. et al., 2023; Soldani et al., 2021; Pulnil and Senivongse, 2022)
Endpoint-Based Service Interaction	µTOSCA, Arcan, KubeFreshener	Redesign services to minimize direct endpoint calls, which can create tight coupling and reduce flexibility. Adopt messaging patterns like event-driven architecture or message queues to decouple services, improving scalability and fault tolerance	(Soldani. et al., 2023; Soldani et al., 2021; Neri et al., 2020)
ESB Usage	Arcan, MSANose	Replace the centralized Enterprise Service Bus (ESB) with decentralized communication methods, such as direct service-to-service communication or lightweight messaging systems	Taibi and Lenarduzzi (2018); Pulnil and Senivongse (2022)
Hard-Coded Endpoints	Arcan, MARS, MSANose	Remove explicit endpoint references from service code and adopting dynamic or configuration-based approaches for endpoint discovery	Taibi and Lenarduzzi (2018) Pulnil and Senivongse (2022)
Hub-like Dependency	Arcan	Restruct services to reduce their reliance on a central hub for communication. This includes promoting direct interactions between services and decentralizing communication patterns	
Inappropriate Service Intimacy	Arcan, MSANose	Adjust service boundaries to reduce unnecessary dependencies and interactions between services	Taibi and Lenarduzzi (2018)
Microservice Greedy	Arcan, MSANose	Consolidate overly granular microservices that serve limited purposes into larger, more cohesive units	Taibi and Lenarduzzi (2018) Pulnil and Senivongse (2022)
Nano Service	Arcan, MAIG, MARS	Consolidate excessively small microservices, which perform trivial or minimal functions, into larger, more cohesive units	
No API Gateway	µTOSCA, Arcan, Kubefreshener, MARS, MSANose	Introduce an API gateway to serve as a central entry point for client requests, providing various functionalities such as routing and authentication	Soldani. et al. (2023) Neri et al. (2020)
Shared Database	µTOSCA, Arcan, MARS, MSANose	Decouple microservices by migrating from a single, shared database to separate databases per service	Soldani et al. (2021) Neri et al. (2020) Pulnil and Senivongse (2022)
Shared Libraries	Arcan, MARS, MSANose	Break down monolithic shared libraries into smaller, specialized libraries or services that can be independently deployed and maintained	Taibi and Lenarduzzi (2018) Pulnil and Senivongse (2022)
Too Many Standards	Arcan, MSANose	Rationalize and standardize the technology stack, protocols, and development practices	Taibi and Lenarduzzi (2018) Pulnil and Senivongse (2022)
Wobbly Service Interaction	µTOSCA, Arcan, KubeFreshener	Stabilize and optimize communication patterns between microservices to ensure consistency, reliability, and efficiency. Such as introducing timeouts and circuit breakers	Soldani. et al. (2023) Soldani et al. (2021) Neri et al. (2020)
Wrong Cuts	Arcan, MARS, MSANose	Reassess and adjust the boundaries of microservices to better align with business domains and functional responsibilities	Taibi and Lenarduzzi (2018) Pulnil and Senivongse (2022)