Date available online: 23-07-2025
Vol. 3, Issue 1 (January – June 2025)

DOI: https://doi.org/10.24312/ucp-jeit.03.01.515

Microservice Antipatterns: Causes, Detection, and Refactoring Challenges

Junaid Aziz[1], and Ghulam Rasool[1]
[1]Department of Computer Science, COMSATS University Islamabad, Lahore Campus, Lahore 54000, Pakistan

Corresponding author: Junaid Aziz (e-mail: mjunaidaziz@gmail.com).

ABSTRACT
Microservice antipatterns negatively impact quality, necessitating a thorough understanding of their causes, effects, and solutions. This study provides a comprehensive review of antipatterns after analyzing 50 studies through a multivocal literature review. Key findings show that unprepared adoption and team culture are major causes, affecting maintainability, performance, and testing. Detection techniques are categorized into five groups, with most tools using search-based approaches. Four refactoring strategies were identified, along with their limitations. The study also highlights research gaps and challenges, guiding future work in improving detection and refactoring methods to mitigate antipattern effects.

INDEX TERMS: Microservice Architecture; Anti-patterns; Antipatterns Detection; Antipatterns Refactoring

I. INTRODUCTION

Despite the benefits of using Microservice architecture (MSA), there exist several open challenges, which can be grouped into two categories: technological and organizational [1]. Both are critical to the correct functioning of the system. However, these challenges may differ slightly when a large existing code base from monoliths is converted to microservices as compared to creating microservices from scratch. A study performed by Alshuqayran et al. [5] on MSA found communication, deployment operations, security, service discovery, and performance as major challenges of using microservices. Similarly, Jamshidi et al. [3] illustrated not only challenges faced by microservices but also envisioned the development of common microservice infrastructure through industry-academia collaboration to tackle such problems.

Recently, a discussion has emerged about taking a viewpoint of practitioners on the definition of antipatterns along with refactoring techniques and tools proposed in academic literature. Lacerda et al. [4] observed that the knowledge of developers about antipatterns detection and refactoring can not only help to improve tools but also the process of refactoring itself. Tahir et al. [5] also used data from the Stack Exchange website to identify the gap between what researchers and developers discuss about code smells and antipatterns? Based on their observations, they came to several conclusions: such as 1) most of the antipatterns detection tools only provide support for a few popular languages, 2) only developers can evaluate the level of antipatterns in a piece of code. Tian et al. [6] conducted an exploratory study to take the viewpoint of developers on architecture antipatterns by analyzing related discussions on Stack Overflow. Posts related to different architecture styles including microservices were extracted and analyzed. Results of their study indicate that detection and refactoring solutions must consider the causes of architectural antipatterns, and practitioners tend to use static code analysis tools to detect and refactor architectural antipatterns. Additionally, practitioners are concerned about the impact of architecture antipatterns on the performance and maintainability of the system and advocates for further research in this regard [7]. Moreover, they are also facing a lack of tool support in this regard. Considering these factors, it is important to mine the literature and identify all the causes and impact of antipatterns on microservices. This will help the community in building appropriate tools not only for dealing with factors causing antipatterns in microservices but also to address the concerns of practitioners about reducing level of different impacts of antipatterns i.e., performance on microservice-based applications. This multivocal literature review (MLR) tries to fill this gap by consolidating both academic and industrial knowledge. The objective of this MLR is to capture the state of art and practice on microservice antipatterns. The following are the major contributions of this study

- Provide an overview of types, causes, and impact of microservice antipatterns reported by both academia and industry
- Outline techniques and tools employed by both researchers and practitioners for the detection as well as correction of microservice antipatterns
- Bridge the gap between researchers and practitioners by revealing deficiencies in existing techniques and tools along with identifying potential research opportunities.

The remainder of this paper is arranged as follows. In Section 2 related work is discussed. Section 3 outlines the research methodology employed in this study. Section 4 presents the results along with a detailed discussion. Finally, Section 5 provides the conclusions drawn from the research.

II. RELATED WORK

There have been few attempts aiming at reviewing the state-of-the-art and current practices on microservice antipatterns. An overview of these studies is illustrated here along with a summary which is shown in Table 1.

Mumtaz et al. [9] performed a mapping study to

discuss different architecture antipatterns detection techniques and tools. They not only observed the lack of tools but also highlighted the need for the identification of software metrics and their thresholds for detecting microservice antipatterns. As per their recommendations, the applicability of these techniques and tools should be based on the software development industry's perspective. They emphasize doing empirical validations with real-world projects spanning many areas and programming languages in this regard.

Taibi et al. [10] identified several agreed microservice architectural patterns widely adopted and reported advantages along with disadvantages for each pattern. In their study, they pointed towards different emerging issues such as the impact of an increase in the number of microservices on the quality of the system, choice of most suitable DevOps tool, the existence of antipatterns, etc.

Ponce et al. [11] conducted MLR and grouped security antipatterns based on different properties such as confidentiality, integrity, and authenticity. They also presented a taxonomy of these antipatterns along with refactoring. However, a proposition of a tool capable of automatically detecting and refactoring security antipatterns in microservice-based applications is lacking in their work. Besides, empirical validation of their proposed refactoring solutions is also missing in their work.

Table 1: Summary of related systematic literature reviews

Study	Studies Reviewed	Study Focus	Search Period	Study Type
[9]	85	Detection of architecture antipatterns	1999-2019	SMS
[10]	42	Advantages and disadvantages of microservice patterns	2014-2017	SMS
[11]	58	Security antipatterns and refactoring	2014-2020	MLR
[12]	31	Visualizing Anti-Patterns in Microservices at Runtime	Not specified	SMS

Abbreviations: MLR, Multivocal literature review; SLR, Systematic literature review; SGLR, Systematic grey literature review; SMS, Systematic mapping study; TLR, Tertiary literature review.

In a mapping study performed by Parker et al. [12], it is analyzed how anti-patterns in microservices can be visualized from a dynamic perspective. Based on the findings, a gap between visualization and detection of microservice antipatterns is highlighted. It is also found that among all available tools proposed in academic literature, no single tool is completely capable of detecting and visualizing them. However, their study is lacking the analyses performed on tools contributed from the industry such as Jaeger [13] and Zipkin [14].

In this study we intend to identify causes as well as the impact of microservice antipatterns. This information will help researchers and practitioners in automating the process of detecting and refactoring microservice antipatterns as suggested by Tian et al. [6] and Aziz et al. [8]. Besides, information about techniques and tools currently applied for detection as well as refactoring of microservice antipatterns is synthesized. Additionally, the limitations of such techniques and tools are also highlighted.

III. RESEARCH METHODOLOGY

This Multivocal Literature Review (MLR) was conducted following the guidelines established by Garousi et al. [15], which are derived from the Systematic Literature Review (SLR) methodology proposed by Kitchenham et al. [16]. In accordance with these guidelines, the MLR process consists of three key stages: planning, conducting, and reporting the review. Figure 1 illustrates the steps involved in each stage. During planning stage, initially we set a goal for this study which is to capture the state of the art and practices in identifying types, causes, impact, detection, and refactoring techniques used for microservice antipatterns. Then, to make a decision about including grey literature in this study, a questionnaire is set (see Table 2) with possible answers either Yes, Maybe or No as per the guidelines by Garousi et al. [15]. Answer to each question is provided by authors through consensus. After consolidating the results, majority of questions are found to be responded in yes. This led us to the need of conducting a comprehensive MLR instead of a Systematic literature review to find answers to the following research questions:

- **RQ1:** What are the main causes that lead to antipatterns in microservices?
  Rationale - We want to explore the causes of antipatterns in microservices reported by researchers and practitioners.
- **RQ2:** How do antipatterns affect microservices?
  Rationale - We want to study the impact of antipatterns on microservice-based applications specifically on the process, performance, and people.
- **RQ3:** What techniques and tools are used for detecting antipatterns in microservices?
  Rationale - We want to learn about techniques and tools that are used by researchers and practitioners for the detection of antipatterns in microservices.
- **RQ4:** What are the refactoring techniques currently employed to resolve antipatterns in microservices?
  Rationale - We want to discover refactoring solutions proposed by researchers and practitioners to mitigate the effects of antipatterns in microservices.

During conducting stage, first, search string and data sources for academic and grey literatures were finalized as shown in Table 3. Then, authors conducted search for relevant academic and grey literature studies using respective data sources. The final selection of studies was made with consensus whereas conflicts were resolved through the mediation of another researcher. Following steps were performed for the search and selection of primary studies:

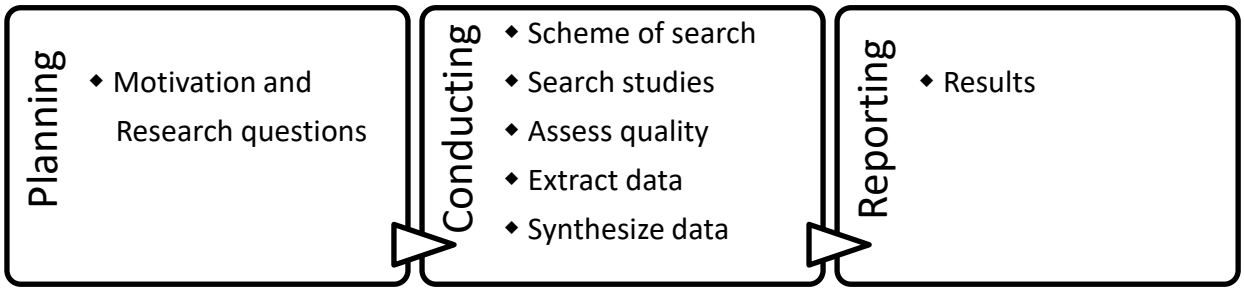

Figure 1: Steps followed for MLR

Table 2: Questionnaire used for including grey literature

Question	Response
Is academic literature not enough to provide solution for the research problem?	Yes
Is quality of evidence generated from academic literature lacking?	Maybe
Is finding context of the research problem in relation to practice necessary?	Maybe
Is this an attempt to validate research outcomes with experiences of practitioners or vice versa?	Yes
Is this an attempt to support research findings with practical experiences?	Yes
Is insights gained from academic and grey literature studies useful for one or both communities?	Yes
Is there an interest shown by practitioners in research problem through large number of contributions?	Yes

Step1 — Search process: For academic literature when we ran the search string, it provided 1032 results whereas for grey literature (see Table 3), it yielded 190,000 results on Google and 101,000 on Bing as shown on top of results page. DuckDuckGo was not providing this information on its results page. Initially, we limited our review of these results by title and abstract to the first 10 pages on every search engine. Afterwards, we gradually moved to the results on other pages until we found that at least half of the results on a page were not pertinent for this research. Duplicate results were also discarded at this stage.

Step2 — Quality assessment: The quality of the selected academic studies was assessed using the formula (1) opted by Ahmad et al. [17]. This formula is based on the recommendations presented in [18] for the qualitative assessment of selected studies. To calculate the quality score, the formula uses five general (i.e., QA1 to QA5) and five specialized assessment elements (i.e., QA6 to QA10) mentioned in Table 5. Since specific contributions of a study are more important than general factors for assessment, therefore, they are assigned 75% weight. An academic study was included if its accumulative quality score was greater than or equal to 1.5.

$$Quality\ Score = \left[\frac{\sum_{G=1}^{5}}{5} + \left(\frac{\sum_{S=1}^{5}}{5} \times 3\right)\right] \quad (1)$$

The quality of grey literature was assessed using the criteria suggested by Garousi et al. [15] as specified in Table 4. Every item of the criteria was assessed one by one for each study by the authors.

After consolidating the results, a grey literature study with a score of 8 or more (set through consensus) was included in the final list of primary studies with decent quality and rest were excluded. This provided us 33 academic (see Table A.2 in Appendix A) and 15 grey literature studies (see Table A.1 in Appendix A). After performing forward and backward snowballing on these studies, 2 further studies were found only for academic literature. We collaboratively extracted and encoded the necessary data from each of the selected primary studies using open and selective coding [69]. Initially, we extracted the metadata such as name, publication year, publication type (for academic literature) and contribution type (for grey literature). Table 6 provides a precise view of a complete list of defined metadata used in this study.

Table 3: Search string and data sources used in this MLR

Literature	Data Source	URL	Search String
Academic	IEEE Xplore	https://ieeexplore.ieee.org	(smell OR antipattern OR anti-pattern OR debt OR anomal OR refactor OR fault OR challeng OR vulnerab) AND (microservice OR micro-service)
	ACM Digital Library	https://dl.acm.org	
	Springer	https://link.springer.com	
	ScienceDirect	https://www.sciencedirect.com	
	DBLP	https://dblp.org	
	Scopus	https://www.scopus.com/	
Grey	Google	www.google.com	
	Bing	https://www.bing.com	
	DuckDuckGo	https://duckduckgo.com	

Table 4: Quality assessment for grey literature

Code	Items (Yes = 1 , No = 0)
QA1	Is this a reliable publisher or is the author belonged to a credible organization?
QA2	The author has published in the subject before
QA3	The author is an expert in the field
QA4	The claim of the sources is accurate
QA5	There are no ulterior motives
QA6	The data backs up the conclusions
QA7	The source has a defined goal
QA8	The source addresses a specific problem (s)
QA9	The methodology used by the source is explicitly explained
QA10	The source includes references to support the claims stated in the research
QA11	Limitations are clearly stated
QA12	The date of the source is clearly stated
QA13	The source talks about or links related GL or formal sources
QA14	The source enriches or adds something to the field of microservice antipatterns
QA15	The source supports or contradicts a current assertion
QA16	To support the claims presented in the study, the source includes citations and backlinks

Table 5: Quality assessment for academic literature

Code	Items (Yes = 1, Partial = 0.5, No = 0)
General	
QA1	Study provides problem definition and motivation
QA2	Research environment is clearly explained
QA3	Research methodology is presented in the study
QA4	Insights and lessons learned are explicitly mentioned
QA5	Contributions along with results are explicitly discussed
Specific	
QA6	Research focus is clearly on antipatterns in microservices
QA7	Study gives a clear picture of problems, solutions, and challenges concerning antipatterns in microservices
QA8	Research clearly states the validation technique applied on its outcome and relevant threats
QA9	Research presents techniques and tools used for the detection or correction of antipatterns in microservices
QA10	Study identifies new directions related to antipatterns in microservices

Table 6: Data Extraction Form: A = Academic Literature, G = Grey Literature

Search Criteria	Data Item	Description	Source
Demographic info	Study ID	Reference Code (AL, GL) sequentially incremented	A/G
	Name	Study title	A/G
	Publication Type	J=Journal, C=Conference	A
	Study Aim	Personal notes about each study	A/G
	Year	Publication year	A/G
	URL	URL of publication	A/G
	Contribution Type	Blog post, Industrial whitepaper, Video, Article, Book	G
RQ1	Causes	Causes of antipatterns	A/G
	Study Design	Experimental, Empirical Study, Solution proposal, Case Study, Personal experience, Tool	A/G
RQ2	Impact	Impact of antipatterns	A/G
RQ3	Detection of antipatterns	Techniques, tools applied for detection of antipatterns in microservices	A/G
RQ4	Refactoring	Solutions to resolve antipatterns in microservices	A/G

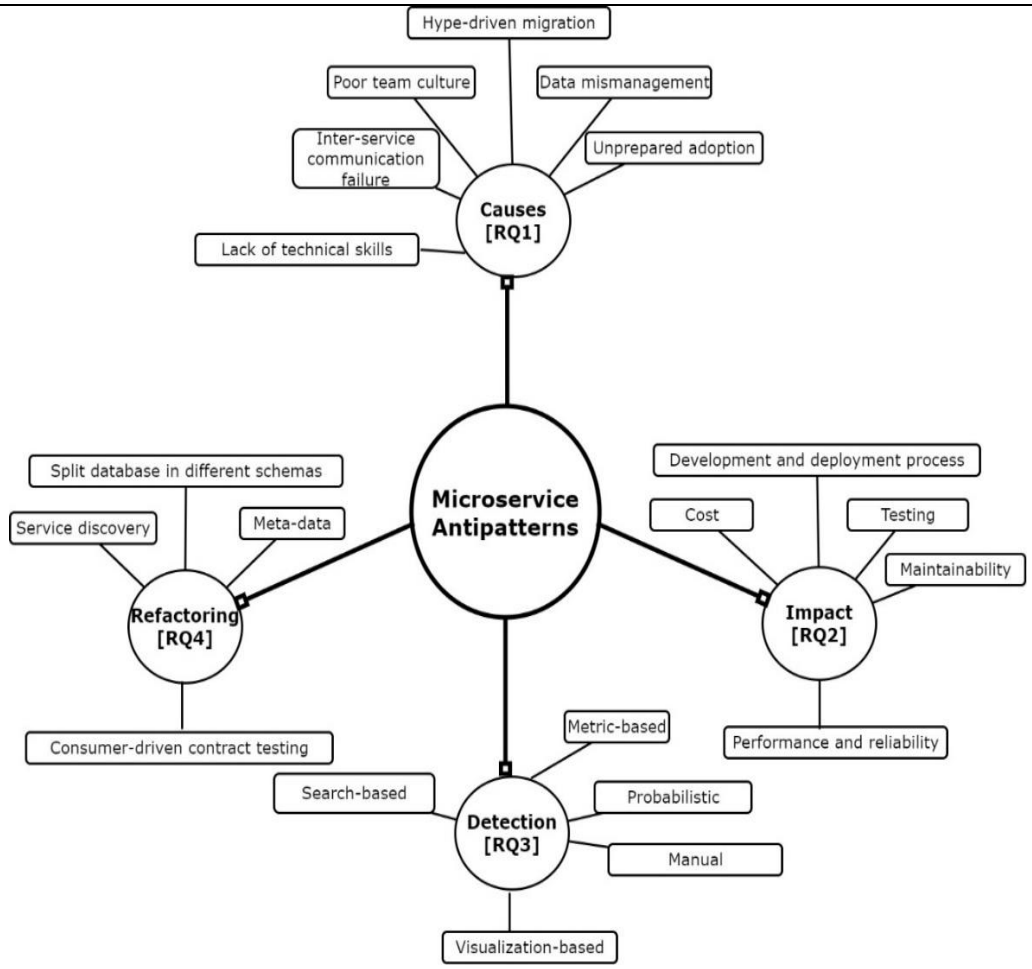

Figure 2: Overview of the study

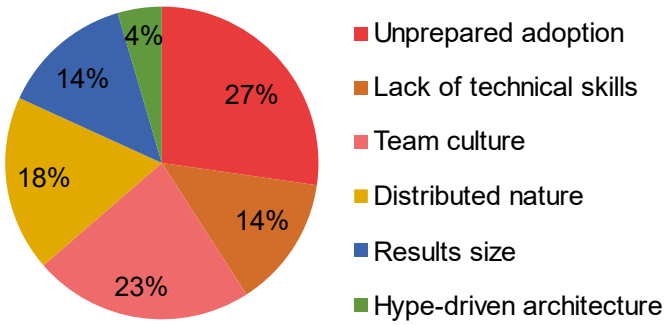

Figure 3: Causes of microservice antipatterns

IV. RESULTS AND DISCUSSION

This section presents and discusses the key findings derived from evaluating and synthesizing the selected studies in relation to each research question. Figure 2 summarizes the key findings of our RQs.

A. RQ1: What are the main causes that lead to antipatterns in microservices?

This study has identified various reasons that introduce antipatterns in microservices. These reasons are extracted from the studies based on industrial surveys and experiences gained from migration of legacy systems to MSA. We classified them in a list below and also summarized them in Figure 3. From the list, some of the causes are found to be overlapping with the causes of antipatterns in other type of applications as mentioned in [70]. However, distributed nature and hype-driven architecture are found applicable only to MSA.

- **Unprepared adoption:** Before deciding to migrate to MSA, organizations need to think about the extra effort required to work on automated deployment, monitoring, failure, eventual consistency, and other issues that this architecture style introduces. Many authors (A1, A5, A7, A21, G3,G13) have mentioned this cause in their studies.
- **Lack of technical skills:** Microservices adoption is

difficult since it demands knowledge of new techniques and technologies, as well as the requirement to automate software deployment and monitoring operations (A7, A9, A22). Therefore, the development team with poor technical skills and their lack of awareness of the tools involved can easily cause antipatterns in such types of applications as highlighted in (A32).

- **Team culture:** Many authors (A5, A7, A20, G3, G13) have cited practices and culture that exist in teams or organizations as a cause of antipatterns in microservices. For instance, teams must be organized to handle microservices independently. Otherwise, this may lead to the development and deployment-related issues.
- **Distributed nature:** The current literature (A3,A18,G4,G15) indicates that MSA implies some challenging problems of data integrity, management of microservices, and their consistency due to its distributed nature. These challenges can lead to different types of antipatterns in applications.
- **Results size:** In (A18, A20, G4), the authors highlighted the need for an efficient and consistent database management solution, especially in data-driven microservices applications that process large amounts of data. According to them, this can seriously affect the performance of the software.
- **Hype-driven architecture:** Shifting to MSA because of its popularity only, believing that all your software-related problems will be solved may cause antipatterns and affect the quality of the software (A22).

The implementation of microservices require that organizations have empowered small teams handling them in a way that corresponds to particular business domains (A5). Product owners, architects, developers, quality engineers, and operational engineers are just a few of the roles that these teams must include in order to provide these services. Complicated code bases are typically produced by large teams, which hinders and delays future updates (A21, G3). In addition, teams that lack authority frequently face delays while they wait for higher-ups to make decisions. Similarly, a lack of alignment with business goals results in dependencies between teams, leading to more delays and drifting away from MSA. It is crucial for teams to possess essential skills like API design, development, and understanding of distributed applications (A7). Without these skills, there might be increased costs for training or hiring, potentially diverting the solution from the microservices approach as teams fall back on their familiar technologies (A9).

The culture within an organization significantly influences its approach to working on systems, as it shapes the individual decisions made by its members. In project-centric cultures, teams are assembled to tackle specific issues, disbanding once the problem is resolved. However, this practice often results in a loss of valuable knowledge about both the problem and its solution. Additionally, if there is a need to revisit the same problem or make further modifications to a component, reconstructing the team or recovering lost knowledge can pose challenges. In the case of microservices, it becomes quite difficult for an organization if it decides to operate in project-centric culture instead of offering team-based ownership of components (A20, G3). Similarly, whenever outsourcing is conducted; the outsourced team can neither adopt the desired organizational culture as it is nor it can avoid such change. This suggests that culture plays a crucial role in determining the suitability of companies or individuals selected to endorse the outsourcing model.

Implementation of MSA brings a lot of challenges and it doesn't work for every organization. Some organizations may find it the only way to keep up with rapid development and deliver software on time. In general, organizations that mismatch any of these causes which are discussed here will pay a toll in the form of antipatterns when attempting to apply MSA. So, instead of just following the hype, the decision about such a transition should be made after evaluating its cost and gains (A22).

B. RQ2: How do antipatterns affect microservices?

Antipatterns in microservices can harm the resulting applications if suitable techniques and processes are not followed. Different authors have mentioned those impacts which are shown in Figure 4.

Microservices need to be as autonomous and decoupled as possible to ease the development and deployment process (G14). This implies that using norms and standards for the definition of microservices contracts can make a huge impact on resulting applications (A13, A19).

Centering microservices around technical concerns only can easily mutate into something called layering which was the main disadvantage of service-oriented architecture. This can eventually incur performance and reliability issues such as high network latency, failures of one service affecting others, and slowing down the whole development process (A1, A3, G6). In (A31), authors performed an experimental study to validate the existence of correlation between microservice antipatterns and microservices performance. Based on their findings, Cyclic Dependency and Shared Persistence have significant negative effect on the performance of microservices. More such studies are needed to find out impact of different microservice antipatterns on metrics such as performance, maintenance, cost etc.

Developers frequently construct test cases (e.g., unit, integration, and system test cases) without having a complete understanding of the operational environment or user behavior. Faults are typically identified only during the process of updating from one service version to another, or occasionally after the service update has been completed (A23, A46, G1).

Implementation of MSA requires organizations to adopt DevOps. Lack of such practice can cause different forms of antipatterns such as API Versioning, Human Evolvability, and Lack of evaluation methods. These antipatterns will eventually hinder the maintainability of the system (A19, A24, A32, G6). Furthermore, numerous instances of a service might be active at the same time. Using virtualization to deploy them isn't cost-effective,

and it also adds a lot of processing overhead (A19).

The microservices approach offers significant benefits, but can be expensive as it requires different infrastructure. Also, additional code is often required to communicate between services in the form of API calls. Bad antipatterns such as Cyclic Dependency that are introduced in such calls can make a great impact on the development and deployment of these services (A13). Besides, systems such as service directories, messaging, and queuing services are required to identify appropriate services and then route calls to them. The presence of antipatterns in any of these can affect performance and reliability (A1,A3,A34,A35). Since microservices can be scripts, containers, or entire virtual machines and require a structured way to package and deploy them, the introduction of antipatterns like Red Flag can make a huge impact on them (A24, G6).

Applications based on MSA also need to implement a system for monitoring service performance and behavior, along with specific error handling techniques. If a microservice is not responding then there is no easy way for other services to understand the error or determine the problem. Additional code and monitoring are required to ensure that problems are treated as errors rather than simply piling them up (A23). A mechanism to decide when to include features in services and when to split features into separate services is also required. Otherwise, these services will be affected by antipatterns like Wrong Cut (A46, G1). Moreover, it has been found that the impact of antipatterns on the cost of developing microservices has been studied in (A19) only. Further research in this area especially providing cost comparison of deployment of the microservices-based application on different cloud containers will be effective and helpful.

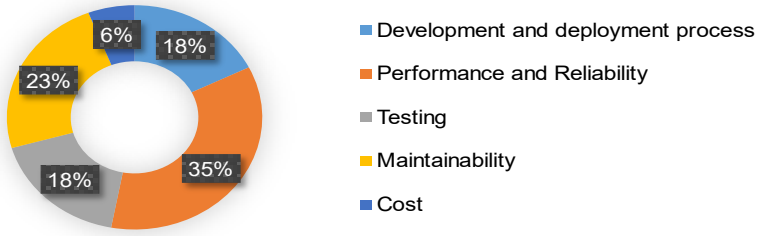

Figure 4: Impact of microservice antipatterns

C. RQ3: What techniques and tools are used for detecting antipatterns in microservices?

Based on the literature review, we have identified techniques used for the detection of antipatterns in microservices. These techniques are further classified into five broad categories (i.e., manual approaches, metric-based approaches, probabilistic approaches, visualization-based approaches, and search-based approaches) as proposed by Kessentini et al. [71]. The list of antipatterns detected by respective category along with the information about tools used for detection is shown in Table 7.

- **D1: Search-based approaches:** Source code and/or bytecode analysis are used to create a realistic representation of the application. This includes a tree representation, identification of the system's endpoints, and the creation of a communication map. Antipatterns in microservices can be detected with the help of these representations. With the use of performance monitoring data, some of the techniques in this category use various machine learning algorithms to classify the actions of the target system. In the first phase, fault injection is used to collect samples of labeled performance data reflecting various service behavior, and multiple classification models are trained using this data. In the second phase, real-time performance data is transferred to these models for the accurate detection of anomalies.
- **D2: Metric-based approaches:** Typically, this type of approach revolves around building an architectural model by employing the reverse engineering approach on source code and communications logs. This model is then used to evaluate the architecture design based on identified principles with corresponding metrics. A dependency graph or other methods can be used to visualize the results.
- **D3: Visualization-based approaches:** This type of approach for detecting antipatterns is about collecting all service invocation links and constructing a service dependency graph representing them. A visual representation of the graph is generated that allows users to browse all service dependency relationships and check for anomalies or errors.
- **D4: Manual approaches:** The information stored in a microservice catalog helps teams not only resolve their production incidents quickly but also build reliable and more operable microservices. It helps to track all the services and systems running in production. Without changing the user code, data regarding resource usage, performance counters, power consumption, and network performance is collected. The information is then utilized to evaluate microservice-based systems in terms of their interactions with the outside world as well as internal connections and dependencies.
- **D5: Probabilistic approaches:** Service workload is continuously monitored and its response time is regularly compared with baseline response time. After applying standard statistical outlier detection techniques, a higher deviation between these two means that the service is suspected to have performance anomalies.

In academic literature, search-based approaches using static code analysis were found to be effective as compared to other techniques because these helped researchers detect a large number of antipatterns in microservices. Besides, tools used for detection have also been made available by them. Moreover, metrics-

based and visualization-based approaches have been experimented on the detection of diverse antipatterns unlike others (i.e., manual and probabilistic approaches) which have been applied for the detection of monitoring type of antipatterns only. In grey literature, microservice catalog is the only technique from the category of manual approaches found to be effective for the detection of a limited number of antipatterns in microservices. Big companies such as LinkedIn, Spotify, Shopify, Bell, etc. have relied on in-house built catalogs but have not made them available online.

D. RQ4: What are the refactoring techniques currently employed to resolve antipatterns in microservices?

Refactoring is a technique that is applied to reorganize the structure of the application without altering its original behavior. It is often performed to remove antipatterns and improve design quality. Even though refactoring is now a common practice in the industry, manual refactoring of antipatterns is still a risky and error-prone task, especially when it is performed by inexperienced people in a team.

Table 7: Antipatterns detection techniques and tools

Technique	Reference Study	Tool Name	Release Type	Languages	Accuracy (%)	Antipatterns Detected	Validation Set
D1	A9	-	-	-	-	LEM	-
	A14	TraceAnomaly	Open Source	Java	P=98, R=97	TA	792 Ms
	A16	-	-	-	-	TA	-
	A25	MEPFL	Experimental	Java	Average P=89 Average R=82	TA	49 Ms
	A26	ARCAN	Prototype	Java	P=100	CD,SP,HE	40 Ms
	A30	MSANose	Open Source	Java	Not measured	WC,NG,SG,CD,SL,LTS, SV,SP, ISI,EU,HE	45 Ms
	A33	MARS	Open Source	Java	Average P=82 Average R=89	WC,NG,CD,SL,SV,SP,HCE	171 Ms
D2	A4	-	-	-	-	SG,CD	-
	A15	-	-	-	-	TA	-
	A27	-	-	-	-	LM	-
D3	A2	-	-	-	-	NST,CD	-
	A6	-	-	-	-	CD,DM,IS	-
D4	G10	-	-	-	-	LTS,LG,HE	-
	G12	-	-	-	-	CD,SP	-
	A34	DEEP-mon	Open Source	Golang,C++	Not measured	LM	36 Ms
D5	A17	-	Experimental	-	Not measured	TA	-

Abbreviations: SP, Shared Persistence; HE, Hard-coded Endpoints; DM, Distributed Monolith; TA, Trace anomaly; CD, Cyclic Dependency; LEM, Lack of evaluation methods; WC, Wrong Cuts; NG, No-API Gateway; SG, Microservice Greedy; SL, Shared Libraries; LTS, Too Many Standards; SV, API Versioning; ISI, Inappropriate Service Intimacy; EU, ESB Usage; LM, Lack of monitoring; NST, No Service Template; IS, Influential Service; LG, Lack of guidance; HE, Human Evolvability; Ms, Microservices; P, precision; R, recall.

Moreover, assigning resources to perform refactoring for microservice antipatterns is not always feasible, due to constraints in the budget, shorter release cycles, and staff shortage. Therefore, researchers and practitioners use different strategies to automate the refactoring process. After reviewing the primary studies, the authors identified the following refactoring strategies:

- **R1: Split database in different schemas:** In (A28, G2, G5) authors have found this refactoring useful for resolving antipatterns like Shared Persistence and Distributed Monolith. This approach can be applied to different situations especially when services access the same data store. For instance, in a scenario where a portion of the data store is accessed by just one service, one way out is to split it into two different data stores, with one storing the portion of data accessed by that single service and the other storing the rest of the data.
- **R2: Consumer-driven contract testing:** In this technique, services communicate with one another using contracts that the consumer creates and then shares with the provider for verification. In most cases, the contract defines a series of transactions between the consumer and the provider. This type of testing technique has proven to be effective in evaluating service integrations. Consumer-driven contract testing, unlike end-to-end testing, can catch all types of errors because it is always done in isolation from other services (A10, G8, G9, G11). The occurrence of antipatterns like Oracle Problem and Test Endpoints can be avoided with the adoption of this strategy
- **R3: Meta-data:** Many authors (A1, A12, A23, A29) have made use of this approach in the form of data flow diagrams, data structures, tests derived from service operation data, microservice usage data,

etc. to resolve antipatterns like Wrong Cuts, Test Endpoints, and Oracle Problem.

• **R4: Service Discovery:** Antipatterns like Hard-coded Endpoints can be resolved with the help of this refactoring which normally occurs in an application when a service A directly invokes another service B either because the location of B is hardcoded in the source code of A, or no message router is used (A28, G7). It may also be possible to dynamically resolve the endpoint of service by simply adding a service discovery mechanism.

Table 8 summarizes the list of techniques that have been applied by researchers and practitioners for refactoring appropriate antipatterns in microservices. Only studies that have provided information about tools either implemented or used are shown in this table. Despite these efforts, it is found that refactoring microservice antipatterns is still at an infancy level. Authors have also witnessed that more interest from the community is shown toward refactoring test antipatterns. Newman [3] also highlight the challenges of performing end-to-end testing on microservices.

In (G11), the author analyzes different types of testing used for microservices and based on personal experience suggests contract testing as a suitable choice. Besides, based on lessons learned from a case study (A10), the authors suggest that consumer-driven contract testing is a feasible practice, especially when dealing with microservices-based applications. Other approaches proposed in (A23, A29) which make use of run-time data of microservices may also help refactor test antipatterns. Refactoring approaches to resolve antipatterns violating key design principles of microservices such as horizontal scalability, isolation of failures, and decentralization is presented in (A28). A prototype is also implemented based on the methodology proposed with limited experimentation.

Table 8: Antipatterns refactoring techniques and tools

Reference	Refactoring Technique				Antipatterns Resolved					Tool Name	Release Type
	R 1	R2	R3	R4	SP	HE	DM	OP	TE		
A28	✓			✓	✓	✓	✓			µFreshener	Prototype
G2	✓						✓			Gremlin	Commercial
G8		✓						✓	✓	Postman	Commercial
A11		✓						✓	✓	Pactflow	Open Source /Commercial
A23			✓					✓	✓	ExVivoMicroTest	Prototype

Abbreviations: SP, Shared Persistence; HE, Hard-coded Endpoints; DM, Distributed Monolith; OP, Oracle Problem; TE, Test Endpoints.

E. Emerging Challenges and Research Opportunities

The study of microservice antipatterns remains an emerging and rapidly evolving research area. Through this investigation, we have identified following key challenges that present opportunities for future research and practical implementation:

• **Many antipatterns still need detection**

Our investigation uncovers that microservice antipatterns manifest not just during development phases, but can also become institutionalized at the organizational level in the absence of effective policy frameworks. The current list of microservice antipatterns detection tools have been developed with a focus on a limited number of antipatterns. Exposure to diverse antipatterns for such tools is needed. Moreover, current and new tools are required to be evaluated on medium to large scale industrial-based microservice systems as it is revealed in this study that the presence of bad antipatterns can impact on performance, maintainability, and testability of microservices.

• **Need to apply other techniques for detection of antipatterns**

This study finds that currently available antipatterns detection tools have mostly made use of search-based techniques only. Researchers need to explore other techniques for antipatterns detection and build corresponding tools. In this regard, metric-based and visualization-based techniques can be experimented with in detecting diverse types of antipatterns. This will also provide an opportunity for finding a more appropriate one, once the results of applying different techniques become available.

• **More research on identifying impact of bad antipatterns on microservice quality needed**

Initial studies suggest a potential relationship between the presence of certain microservice antipatterns and the overall quality of a microservices-based system (A31). However, to establish a comprehensive understanding, further research in this direction is needed. Investigating and quantifying this correlation can provide valuable insights for researchers and practitioners aiming to enhance the design and maintainability of microservices-based systems.

V. CONCLUSION

This study offers a concise yet thorough review of microservice antipatterns by analyzing academic and industry literature between 2014 and 2023. Key findings include:

• Identifying 6 root causes behind antipattern occurrences.
• Assessing their impacts on microservices.
• Classifying detection methods into 5 groups (manual, metric-based, probabilistic, visualization-based, and search-based).
• Highlighting research gaps and opportunities.
• Discovering 4 refactoring approaches for mitigation.

Current research primarily focuses on architecture, design, and organizational antipatterns, with limited tools available. Besides most tools detect only specific type of microservice antipatterns. Additionally, proposed refactoring methods often lack real-world testing.

A comprehensive, multi-language detection tool remains an unmet need in the field.

Appendix A. Studies selected for this MLR

Table A.1: Selected studies in grey literature

ID	Quality Assessment Codes (QA)																Quality score	Reference
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
G1	1	0	0	0	1	0	1	0	1	0	0	1	0	1	1	1	8	[49]
G2	0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	9	[50]
G3	1	1	0	0	1	0	1	0	0	1	1	1	1	1	1	1	11	[51]
G4	0	1	0	0	0	0	1	0	1	1	1	1	1	1	1	0	9	[52]
G5	1	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	10	[53]
G6	0	1	0	1	0	1	1	0	0	1	1	1	0	1	1	1	10	[54]
G7	1	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	11	[55]
G8	0	1	0	1	1	0	0	1	0	1	0	1	1	1	1	0	9	[56]
G9	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	11	[57]
G10	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1	12	[58]
G11	0	1	0	0	1	0	1	0	1	1	0	1	1	1	0	1	9	[59]
G12	1	0	0	1	0	0	1	0	1	1	0	1	1	1	0	0	8	[60]
G13	1	0	0	1	0	1	1	1	1	1	1	1	1	1	0	0	11	[63]
G14	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1	9	[64]
G15	0	1	1	0	0	1	1	0	1	1	1	0	0	1	0	0	8	[68]

Table A.2: Selected studies in academic literature

ID	Title	Year	Quality score	Reference
A1	Challenges of Production Microservices	2018	2.6	[19]
A2	Graph-based and scenario-driven microservice analysis, retrieval, and testing	2019	2.9	[20]
A3	Microservice Disaster Crash Recovery: A Weak Global Referential Integrity Management	2020	3.2	[21]
A4	Evaluation of Microservice Architectures: A Metric and Tool-Based Approach	2018	2.6	[22]
A5	Exploring the Microservice Development Process in Small and Medium-Sized Organizations	2020	2.9	[23]
A6	Service Dependency Graph Analysis in Microservice Architecture	2020	3.4	[24]
A7	An Experience Report from the Migration of Legacy Software Systems to Microservice Based Architecture	2019	2.7	[25]
A8	Tool Support for the Migration to Microservice Architecture: An Industrial Case Study	2019	2.6	[26]
A9	Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring	2018	3.5	[27]
A10	Consumer-Driven Contract Tests for Microservices: A Case Study	2019	3.2	[28]
A11	Fine-Grained Access Control for Microservices	2018	3.3	[29]
A12	From Monolith to Microservices: A Dataflow-Driven Approach	2017	3.5	[30]
A13	Functional-First Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience	2019	3.3	[31]
A14	Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks	2020	3	[32]
A15	Self-Adaptive Root Cause Diagnosis for Large-Scale Microservice Architecture	2020	3.5	[33]
A16	An Intelligent Anomaly Detection Scheme for Micro-Services Architectures With Temporal and Spatial Data Analysis	2020	2.6	[34]
A17	RAD: Detecting Performance Anomalies in Cloud-based Web Services	2020	3	[35]
A18	Framework for Interaction Between Databases and Microservice Architecture	2019	2.1	[36]
A19	Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture	2016	3.6	[37]
A20	An Expert Interview Study on Areas of Microservice Design	2018	3.3	[38]
A21	Migrating Towards Microservice Architectures: An Industrial Survey	2018	2.7	[39]
A22	An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions	2018	3	[40]
A23	Automatic Ex-Vivo Regression Testing of Microservices	2020	3.1	[41]
A24	Integrating Continuous Security Assessments in Microservices	2017	2.9	[42]

	and Cloud Native Applications			
A25	Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs	2019	2.6	[43]
A26	Towards Microservice Antipatterns Detection	2020	2.5	[44]
A27	Towards a method for monitoring the coupling evolution of microservice-based architectures	2020	3.3	[45]
A28	Freshening the Air in Microservices: Resolving Architectural Antipatterns via Refactoring	2020	2.9	[46]
A29	Testing microservice architectures for operational reliability	2020	3.6	[47]

Table A.2 (continued)

ID	Title	Year	Quality score	Reference
A30	Automated Code-Smell Detection in Microservices Through Static Analysis: A Case Study	2020	3.1	[48]
A31	An Empirical Study on Underlying Correlations between Runtime Performance Deficiencies and “Bad Antipatterns” of Microservice Systems	2021	2.5	[61]
A32	Impacts, causes, and solutions of architectural antipatterns in microservices: An industrial investigation	2022	3.7	[62]
A33	On the maintenance support for microservice-based systems through the specification and the detection of microservice antipatterns	2023	3.9	[65]
A34	Identifying Anti-Patterns in Distributed Systems With Heterogeneous Dependencies	2023	2.6	[66]
A35	An Approach for Evaluating the Potential Impact of Anti-Patterns on Microservices Performance	2023	2.8	[67]

References

[1] Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems (1st ed.). O’Reilly Media.

[2] Alshuqayran, N., Ali, N., & Evans, R. (2016, November). A systematic mapping study in microservice architecture. In 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA) (pp. 44-51). IEEE.

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

[4] Lacerda, G., Petrillo, F., Pimenta, M., & Guéhéneuc, Y. G. (2020). Code antipatterns and refactoring: A tertiary systematic review of challenges and observations. Journal of Systems and Software, 167, 110610.

[5] Tahir, A., Dietrich, J., Counsell, S., Licorish, S., & Yamashita, A. (2020). A large scale study on how developers discuss code antipatterns and anti-pattern in Stack Exchange sites. Information and Software Technology, 125, 106333.

[6] Tian, F., Liang, P., & Babar, M. A. (2019). How Developers Discuss Architecture Antipatterns? An Exploratory Study on Stack Overflow. 2019 IEEE International Conference on Software Architecture (ICSA).

[7] Zhou, X., Li, S., Cao, L., Zhang, H., Jia, Z., Zhong, C., ... & Babar, M. A. (2023). Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. Journal of Systems and Software, 195, 111521.

[8] Aziz, J., & Rasool, G. (2024). A Design-Oriented Classification of Microservice Smells. *UCP Journal of Engineering & Information Technology (HEC Recognized-Y Category)*, *2*(2), 33-40.

[9] Mumtaz, H., Singh, P., & Blincoe, K. (2020). A systematic mapping study on architectural antipatterns detection. *Journal of Systems and Software*, 110885.

[10] Taibi, D., Lenarduzzi, V., & Pahl, C. (2019). Continuous Architecting with Microservices and DevOps: A Systematic Mapping Study. Cloud Computing and Services Science, 126–151.

[11] Ponce, F., Soldani, J., Astudillo, H., & Brogi, A. (2022). Antipatterns and refactorings for microservices security: A multivocal literature review. Journal of Systems and Software, 192, 111393. doi:10.1016/j.jss.2022.111393

[12] Parker, G., Kim, S., Al Maruf, A., Cerny, T., Frajtak, K., Tisnovsky, P., & Taibi, D. (2023). Visualizing Anti-Patterns in Microservices at Runtime: A Systematic Mapping Study. IEEE Access.

[13] Jaeger. (n.d.). Retrieved April 30, 2023, from https://www.jaegertracing.io/

[14] Zipkin. (n.d.). Retrieved April 30, 2023, from https://zipkin.io/

[15] Garousi, V., Felderer, M., & Mäntylä, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, *106*, 101-121.

[16] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software engineering," in "EBSE Technical Report," 2007, vol. EBSE- 2007-01

[17] Ahmad, A., Babar, M.A., 2016. Software architectures for robotic systems: A systematic mapping study. J. Syst. Softw. 122 (12), 16–39.

[18] Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering Domain. J. Syst. Softw. 80 (4), 571–583.

[19] Götz, B., Schel, D., Bauer, D., Henkel, C., Einberger, P., & Bauernhansl, T. (2018). Challenges of production microservices. *Procedia CIRP*, *67*, 167-172.

[20] Ma, S. P., Fan, C. Y., Chuang, Y., Liu, I. H., & Lan, C. W. (2019). Graph-based and scenario-driven microservice analysis, retrieval, and testing. *Future Generation Computer Systems*, *100*, 724-735.

[21] Manouvrier, M., Pautasso, C., & Rukoz, M. (2020, June). Microservice Disaster Crash Recovery: A Weak Global Referential Integrity Management. In *International Conference on Computational Science* (pp. 482-495). Springer, Cham.

[22] Engel, T., Langermeier, M., Bauer, B., & Hofmann, A. (2018, June). Evaluation of microservice architectures: A metric and tool-based approach. In *International Conference on Advanced Information Systems Engineering* (pp. 74-89). Springer, Cham.

[23] Sorgalla, J., Sachweh, S., & Zündorf, A. (2020, November). Exploring the microservice development process in small and medium-sized organizations. In *International Conference on Product-Focused Software Process Improvement* (pp. 453-460). Springer, Cham.

[24] Gaidels, E., & Kirikova, M. (2020, September). Service Dependency Graph Analysis in Microservice Architecture. In *International Conference on Business Informatics Research* (pp. 128-139). Springer, Cham.

[25] da Silva, H. H. S., Carneiro, G. D. F., & Monteiro, M. P. (2019). An experience report from the migration of legacy software

systems to microservice based architecture. In *16th International Conference on Information Technology-New Generations (ITNG 2019)* (pp. 183-189). Springer, Cham.

[26] Pigazzini, I., Fontana, F. A., & Maggioni, A. (2019, September). Tool support for the migration to microservice architecture: An industrial case study. In *European Conference on Software Architecture* (pp. 247-263). Springer, Cham.

[27] Du, Q., Xie, T., & He, Y. (2018). Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring. Algorithms and Architectures for Parallel Processing, 560–572.

[28] Lehvä, J., Mäkitalo, N., & Mikkonen, T. (2019, November). Consumer-driven contract tests for microservices: A case study. In *International Conference on Product-Focused Software Process Improvement* (pp. 497-512). Springer, Cham.

[29] Antonio, N., Vitor, J., Khaled, M., & Ali, A. (2018, October). Fine-Grained Access Control for Microservices. In The 11th International Symposium on Foundations & Practice of Security (Vol. 11358). Springer.

[30] Chen, R., Li, S., & Li, Z. (2017, December). From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 466-475). IEEE.

[31]Gouigoux, J. P., & Tamzalit, D. (2019, March). "Functional-First" Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)* (pp. 182-186). IEEE.

[32] Liu, P., Xu, H., Ouyang, Q., Jiao, R., Chen, Z., Zhang, S., Yang, J., Mo, L., Zeng, J., Xue, W., & Pei, D. (2020). Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE).

[33] Ma, M., Lin, W., Pan, D., & Wang, P. (2020). Self-Adaptive Root Cause Diagnosis for Large-Scale Microservice Architecture. *IEEE Transactions on Services Computing*.

[34] Zuo, Y., Wu, Y., Min, G., Huang, C., & Pei, K. (2020). An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis. *IEEE Transactions on Cognitive Communications and Networking*, *6*(2), 548-561.

[35] Mukherjee, J., Baluta, A., Litoiu, M., & Krishnamurthy, D. (2020, October). RAD: Detecting Performance Anomalies in Cloud-based Web Services. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)* (pp. 493-501). IEEE.

[36] El Kholy, M., & El Fatatry, A. (2019). Framework for interaction between databases and microservice architecture. *IT Professional*, *21*(5), 57-63.

[37] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *Ieee Software*, *33*(3), 42-52.

[38] Haselböck, S., Weinreich, R., & Buchgeher, G. (2018, November). An expert interview study on areas of microservice design. In *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)* (pp. 137-144). IEEE.

[39] Di Francesco, P., Lago, P., & Malavolta, I. (2018, April). Migrating towards microservice architectures: an industrial survey. In *2018 IEEE International Conference on Software Architecture (ICSA)* (pp. 29-2909). IEEE.

[40] Luz, W., Agilar, E., de Oliveira, M. C., de Melo, C. E. R., Pinto, G., & Bonifácio, R. (2018, September). An experience report on the adoption of microservices in three Brazilian government institutions. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering* (pp. 32-41).

[41] Gazzola, L., Goldstein, M., Mariani, L., Segall, I., & Ussi, L. (2020, October). Automatic ex-vivo regression testing of microservices. In *Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test* (pp. 11-20).

[42] Torkura, K. A., Sukmana, M. I., & Meinel, C. (2017, December). Integrating continuous security assessments in microservices and cloud native applications. In *Proceedings of the10th International Conference on Utility and Cloud Computing* (pp. 171-180).

[43] Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., ... & He, C. (2019, August). Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 683-694).

[44] Pigazzini, I., Fontana, F. A., Lenarduzzi, V., & Taibi, D. (2020, June). Towards microservice antipatterns detection. In *Proceedings of the 3rd International Conference on Technical Debt* (pp. 92-97).

[45] de Freitas Apolinário, D. R., & de França, B. B. N. (2020, October). Towards a method for monitoring the coupling evolution of microservice-based architectures. In *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse* (pp. 71-80).

[46] Brogi A., Neri D., Soldani J. (2020) Freshening the Air in Microservices: Resolving Architectural Antipatterns via Refactoring. In: Yangui S. et al. (eds) Service-Oriented Computing – ICSOC 2019 Workshops. ICSOC 2019. Lecture Notes in Computer Science, vol 12019. Springer, Cham.

[47] Pietrantuono, R., Russo, S., & Guerriero, A. (2020). Testing microservice architectures for operational reliability. *Software Testing, Verification and Reliability*, *30*(2), e1725.

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

[49] Mahran, L. (2020). TESTING MICROSERVICES: PRINCIPLES, CHALLENGES, CASE STUDIES [Blog]. Retrieved 2024, from https://mobidev.biz/blog/testing-microservices-principles-challenges-case-studies.

[50] Newman, A. (2020). Is your microservice a distributed monolith? [Blog]. Retrieved 2021, from https://www.gremlin.com/blog/is-your-microservice-a-distributed-monolith/.

[51] Grabner, A. (2016). *Locating Common Micro Service Performance Anti-Patterns*. InfoQ. Retrieved 2021, from https://www.infoq.com/articles/Diagnose-Microservice-Performance-Anti-Patterns/.

[52] Carneiro, C., & Schmelmer, T. (2018). *Microservices from day one*.

[53]Aucion, C. (2018). *How Anti-Patterns Can Stifle Microservices Adoption in the Enterprise | Application Performance Monitoring Blog | AppDynamics*. Retrieved 2024, from https://www.appdynamics.com/blog/engineering/how-to-avoid-antipatterns-with-microservices/.

[54] Dietrich, E. (2018). *Top 4 Ways to Make Your Microservices Not Actually Microservices | Scalyr*. Retrieved 2021, from https://www.scalyr.com/blog/top-4-ways-to-make-your-microservices-not-actually-microservices.

[55] Saleh, T. (2016). *Microservices Antipatterns*. InfoQ. Retrieved 2023, from https://www.infoq.com/presentations/cloud-anti-patterns/.

[56] Postman. (2017). Automated Testing. Retrieved from https://www.postman.com/infographics/automated-testing-whitepaper.pdf

[57] Bulaty, W., & Williams, L. (2019). Testing Microservices: an Overview of 12 Useful Techniques. InfoQ. Retrieved 2021, from https://www.infoq.com/articles/twelve-testing-techniques-microservices-intro/

[58] Laban, J. (2020). Why You Need A Microservice Catalog. Retrieved 2024, from https://www.opslevel.com/2020/04/21/why-you-need-a-microservice-catalog/

[59] Simform, (2019). Microservices Testing Strategies, Types & Tools: A Complete Guide. (2019). Retrieved 2020, from https://www.simform.com/microservice-testing-strategies/

[60] Bogard, J. (2017). Avoiding Microservice Megadisasters [Video]. Retrieved 2020, from https://www.youtube.com/watch?v=gfh-VCTwMw8

[61] Liu, L., Tu, Z., He, X., Xu, X., & Wang, Z. (2021, September). An Empirical Study on Underlying Correlations between Runtime Performance Deficiencies and "Bad Antipatterns" of Microservice Systems. In *2021 IEEE International Conference on Web Services (ICWS)* (pp. 751-757). IEEE.

[62] Zhong, C., Huang, H., Zhang, H., & Li, S. (2022). Impacts,

causes, and solutions of architectural antipatterns in microservices: An industrial investigation. Software: Practice and Experience, 52(12), 2574-2597.

[63] Abbott, M. (2022, August 17). Why is my development team so slow? Retrieved May 1, 2024, from https://akfpartners.com/growth-blog/why-is-my-development-team-so-slow

[64] Bottleneck #01: Tech debt. (n.d.). Retrieved May 1, 2024, from https://martinfowler.com/articles/bottlenecks-of-scaleups/01-tech-debt.html

[65] Tighilt, R., Abdellatif, M., Trabelsi, I., Madern, L., Moha, N., & Guéhéneuc, Y. G. (2023). On the maintenance support for microservice-based systems through the specification and the detection of microservice antipatterns. Journal of Systems and Software, 111755.

[66] Fang, H., Cai, Y., Kazman, R., & Lefever, J. (2023, March). Identifying Anti-Patterns in Distributed Systems With Heterogeneous Dependencies. In 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C) (pp. 116-120). IEEE.

[67] Matar, R., & Jahić, J. (2023, March). 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) (pp. 167-170). IEEE.

[68] SAM, P. D. S. (2023). Principles of Software Architecture Modernization: Delivering Engineering Excellence with the art of fixing microservices, monoliths, and distributed. BPB PUBLICATIONS.

[69] Strauss, A., & Corbin, J. (1990). Basics of qualitative research. Sage publications.

[70] Sharma, T., & Spinellis, D. (2018). A survey on software smells. Journal of Systems and Software, 138, 158- 173.

[71] Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., & Ouni, A. (2014). A cooperative parallel search-based software engineering approach for code-smells detection. *IEEE Transactions on Software Engineering*, *40*(9), 841-861