IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 September 2022, accepted 11 December 2022, date of publication 11 January 2023, date of current version 17 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3236165

TOPICAL REVIEW

Visualizing Anti-Patterns in Microservices at Runtime: A Systematic Mapping Study

GARRETT PARKER[1], SAMUEL KIM[1], ABDULLAH AL MARUF[1], (Member, IEEE), TOMAS CERNY[1], KAREL FRAJTAK[2], PAVEL TISNOVSKY[3], AND DAVIDE TAIBI[4,5], (Member, IEEE)

[1]Department of Computer Science, Baylor University, Waco, TX 76798, USA
[2]Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, 166 36 Prague, Czech Republic
[3]Red Hat, 612 00 Brno, Czech Republic
[4]Empirical Software Engineering in Software, Systems, and Services, University of Oulu, 90570 Oulu, Finland
[5]Cloud and Software Engineering Group, Tampere University, 33720 Tampere, Finland

Corresponding author: Tomas Cerny (tomas_cerny@baylor.edu)

This work was supported in part by the National Science Foundation under Grant 1854049; in part by the Grant from Red Hat Research (https://research.redhat.com); in part by the Grant from the Ulla Tuominen Foundation, Finland; and in part by the Grant from the Academy of Finland under Grant 349488-MuFAno.

ABSTRACT In the world of microservices, companies must be able to create systems that operate in the most efficient way. To achieve this, anti-patterns must be avoided because of their detriment to the quality of the system. Some of the most troubling anti-patterns are hard to detect because of their appearance at runtime. Effectively removing anti-patterns from a system requires dynamic analysis because of the large size of microservice-based systems. While the detection of anti-patterns is helpful, being able to visualize them offers a great benefit to companies working with microservices. Seeing how the overall system is flowing and recognizing the existence of anti-patterns can help improve microservice-based systems. In this paper, a systematic mapping study was performed to find the current state of research on visualizing anti-patterns in microservices from the dynamic perspective. Several hundred papers were examined and a total of 31 were found to be relevant to the research topic. The papers, when analyzed, revealed that there are mechanisms to detect anti-patterns at runtime in microservices, and there are also mechanisms for visualizing the architecture of a microservice-based system. This study's findings could help to identify and remove anti-patterns that occur during runtime in microservices, as well as a means of visualizing these anti-patterns.

INDEX TERMS Anti-pattern, dynamic analysis, mapping study, microservice, visualization.

I. INTRODUCTION

As software systems evolve and change, their analysis must also evolve and change. Microservices are commonly used in web-based systems because of their flexibility and ability to support large scale architectures. While microservices are appealing, they must be designed properly in order to provide the best use for companies. Anti-patterns, also known as bad smells, are common design flaws that can be found across microservices. Understanding and preventing anti-patterns in microservices is a necessity of any company that plans on utilizing microservices [1]. This task is not as easy as it sounds, however, since certain anti-patterns occur at runtime [2] and require dynamic analysis to be discovered. Dynamic analysis can also prove to be costly because of the amount of time it requires. For a microservice-based system to be as efficient as possible, there must be measures taken at runtime to detect and remove anti-patterns.

The reason behind our mapping study was to collect and analyze the current research on how anti-patterns in microservices can be visualized from a dynamic perspective. Tracing through a system at runtime and creating a visual model is a topic that has been discussed across several papers [3], [4], [5]. Combining that visualization aspect with the detection of anti-patterns [6], [7] will allow for system architects to quickly correct design flaws, suggest corrections and improve the quality of the system. As straightforward as this idea may sound, there is a small amount of research

The associate editor coordinating the review of this manuscript and approving it for publication was Mu-Yen Chen.

that has been put towards combining anti-pattern detection and visualization of a microservice–based system at runtime. Although there is little research that combines all aspects of our topic, there is a significant number of papers discussing visualizing microservices and detecting anti–patterns at runtime separately.

This research will be significant to system architects who would like to verify that the system is free of anti-patterns. To be able to automatically discover and detect the location of anti–patterns would be beneficial for many companies utilizing microservice–based systems.

The rest of this paper is organized as follows. Section II defines anti-patterns as well as lists some examples. Section III describes the methods utilized to discover and analyze the existing research on the topic. Section IV presents an analysis of the results after reading through all of the selected papers. Finally, section V concludes the paper with contributions and ideas for future works.

II. BACKGROUND

Patterns are problems that occur often and have solutions that can be applied to them. By utilizing the solutions to problems that have already been faced by others, systems can be designed in the most efficient way. This efficiency is what patterns are typically used for. Many developers repeat the same mistakes made by other developers all over, introducing the same set of previously known issues, anti-patterns, into the system. That makes anti–patterns similar to patterns, but anti–patterns are a detriment to the system [8]. It is important to note that anti-patterns are not errors that result in a system crashing; rather, they are typically design flaws that decrease the quality of service [9]. Many anti-patterns were previously detected and categorized — such as a bottleneck, cyclic dependencies, knots, etc [2]. While these anti-patterns are known and have existed for some time now, one may wonder why they keep resurfacing. Since microservices are used to support large scale systems, only few selected individuals, usually system architects, know how all of the pieces interact. Different people design different sections of a system; one team can be responsible for one microservice without a deep knowledge of the system as a whole. Since they do not see the system as a whole, anti-patterns can manifest without anyone being aware of it. Utilizing a tool that can create a visualization of the system, thus, increases the probability of detecting an anti–pattern.

There are other issues that come with detecting anti–patterns, such as the fact that some anti-patterns manifest in runtime. There has been research put into statically detecting anti–patterns, but static anti-patterns are generally contained in one service and only impact a single aspect of the whole system. To see anti-patterns such as cyclic dependencies, data must be collected at runtime to discover which components of the system communicate with each other [10]. Dynamic analysis is not uncommon when it comes to testing microservices. To ensure that the system is working properly and that logic errors are not occurring, many companies thoroughly test their complete systems in production-like environments [11]. The issue concerning anti-patterns comes from the limitation of testing mechanisms only detecting outstanding errors, which are not caused by anti–patterns. In order to detect anti–patterns at runtime, resources (time and money) must be dedicated to understand the flow of data; otherwise, some of the worst design offenses can go undetected.

It is important to understand the current state of research on microservices because of how quickly systems evolve. We have discovered two mapping studies discussing topics similar to the one presented in this paper. Ponce et al. [32] examined smells in microservices and their impact on security. Although this paper examines smells and their existence in microservices, our mapping study is focused on visualizing anti-patterns and does not examine the security risks that they come with. Bushong et al. [33] present another mapping study that discusses different tools and challenges that come with analyzing a microservice–based system. Though our study mentions analyzing microservices through dynamic analysis, we primarily focus on visualization.

III. MAPPING STUDY METHOD

In this work, we adopted the systematic mapping study methodology, proposed by Petersen et al. [34]. Our complete mapping study document can be found,[1] detailing our filtration and mapping process.

In the first phase, we focused on defining the research questions that would be answered by our study. Our questions went through trial and error throughout the entire process as we wanted to present relevant information about our topic. In the second phase, we drafted a search query that was applied across five research databases. After collecting papers from our query, we moved into the third phase where we began filtering out papers that were unrelated to our topic based on their title and abstract. Once the initial filtering was completed, we moved into the fourth phase where we continued to filter papers after reading the full article. Finally, we concluded with the fifth data analysis phase where we mapped the 31 remaining papers to any research question that it answered.

The questions we examined in this mapping study are as follows:

- RQ1 What methods of detecting patterns in SOA/ Microservices exist? Can these methods be adapted to detect anti-patterns too?
- RQ2 How can dynamic analysis be used to analyze anti-patterns in a system built using microservices?
- RQ3 Which anti-patterns can be visualized in call graphs?
- RQ4 How can visualizing the architecture of a microservice aid in detecting or preventing anti-patterns within that system?
- RQ5 Which tools exist for visualizing anti-patterns through the dynamic perspective?

[1] https://zenodo.org/record/6815837

```
(pattern OR debt OR smell OR degrade)
AND
(microservice OR "cloud-native" OR soa OR "service-oriented architecture")
AND
(visual* OR model OR view OR graph OR interface)
AND
(dynamic OR runtime)
```

Listing 1. Search Query for the Research Databases.

We used five research databases in order to populate our results including: ACM Digital Library (DL), IEEE Xplore, ScienceDirect, Scopus, and SpringerLink. The search query was written to focus our results on papers that discussed the visualization of anti-patterns in microservices from the dynamic perspective. The query was split into four parts. First was *smell* and words related to it such as *pattern, debt,* and *degrade*. The word, ''anti-pattern'', was not included because the term, ''pattern'', collected papers with the word, ''anti-pattern'', and generated more results. Second, *microservice* and terms that describe its predecessors such as *service-oriented architecture,* and *cloud-native*. Third, *visual* and other words that relate to visualization such as *graph, model, view,* and *interface*. Fourth and finally, *dynamic and runtime* to remove papers that only focused on static analysis. The full search query is presented in Listing 1.

After all the papers were collected from the search query, we manually read through every title and abstract to filter papers according to our inclusion and exclusion criteria. We found that many papers were dedicated to maintaining the security of a system and failed to discuss anti-patterns and their detection. Several papers were also solely focused on testing a microservice-based system for correctness and failing to test for anti-patterns. After the initial filtering process, we read through the full text of the remaining papers and excluded those that did not prove to be relevant to our study. If we found a paper related to our problem, we read through the related works to add papers that our query failed to include.

The inclusion criteria we applied is as follows:

1 Papers investigating anti-patterns/patterns as they pertain to microservices/SOA in the visual aspect.
2 Papers that discuss how anti-patterns can be seen from the dynamic perspective of a microservice/SOA.
3 Papers that discuss the visualization of microservices/SOA and mention anti-patternss.
4 Papers which mention how dynamic analysis can be used to analyze anti-patterns within microservices/SOA.
5 Papers that provide use in detecting or preventing anti-patterns in microservice/SOA.

The exclusion criteria we applied is as follows:

1 Papers not written in English.
2 Papers that do not discuss visualization.
3 Duplicates.
4 Opinion papers.
5 Papers that are non-peer reviewed.
6 Papers that do not have the full text available.
7 Papers that do not mentioned neither smells nor detecting issues.

TABLE 1. Search query results for various index sites.

Indexer	Search Results	Filtered	Referenced	Total Relevant
ACM DL	21	2	0	2
IEEE Xplore	87	8	7	15
Scopus	118	5	2	7
ScienceDirect	125[2]	1	1	2
SpringerLink	15[3]	0	0	0
Others	-	-	5	5
Total	332[4]	16	15	31

The results of our search query and filtering process are listed in Table 1 as well as any papers that were added from the references of another paper. After the process was completed, we were left with 31 papers that were thoroughly examined to understand where the current research on visualizing anti-patterns at runtime in microservices stands.

In order to successfully present the research from the 31 collected papers, a map was created to connect each paper with any research question it answered. As we read through the papers, notes were taken on what question a paper answered, as well as how it answered the question. Answering the questions as we read helped us gather the current research presented in section IV and track what information was garnered from each paper. After taking these notes on the papers, we read through all of the answers for a single research question in order to synthesize the information. Using our mapping and the different answers collected from it allowed us to properly gather the information presented in the research papers and ultimately present our findings in this mapping study.

IV. ANALYSIS RESULTS

Of the 332 unique papers that were returned by our search query, only 31 proved to be relevant to our topic. Many papers were either focused on security or testing the system to verify the absence of logic errors. These 31 papers, listed in the Primary studies reference section, were examined and utilized to answer our research questions. In this section, we present the results of our study based on the information gathered from our research.

As mentioned in section II, anti-patterns are similar to patterns as they are problems that have solutions that are applicable to multiple contexts. In a service-oriented architecture (SOA), pattern detection is common because

it allows businesses to verify the quality of a system [12]. Another advantage of this pattern detection is that it is performed at runtime, a necessity for detecting several anti-patterns in microservices. Although the software that is used for pattern detection may be slightly outdated, it is worth examining because of its potential ability to detect anti-patterns.

There are different approaches to pattern detection, but one of the most common tools that we found was complex event processing (CEP) [13], [14], [15]. CEP's power stems from the ability to trace through a large volume of data to recognize patterns. The general method that CEP follows is: the manual input of patterns, recording of events as they take place, and notifying when a pattern has been detected. This manual inputting of patterns allows for the possibility of certain anti-patterns to be described in CEP, and then detected at runtime. CEP can also be connected with visualization tools such that when a pattern is detected and the notification is sent, the visualization tool models it. The power of CEP and pattern detection could provide a mechanism for detecting anti-patterns in modern microservices.

Other papers also mentioned a mechanism for detecting patterns in SOA, and these papers described the process of writing their own algorithm [12], [16]. Although this approach is more complicated than building upon an existing tool, a general algorithm that could be applied across multiple languages would be more beneficial. Gammage et al. [2] introduce an algorithm that utilizes graph theory to detect anti-patterns such as the bottleneck, the knot, and cyclic dependencies. This paper builds upon the existing knowledge of graph elements, such as strongly connected components, to provide a way for detecting anti-patterns. Should the algorithms and knowledge described in this paper be applied to other microservices, many systems can utilize this method to detect and remove anti-patterns.

The main issue that was found regarding pattern detection techniques was that many of the techniques only describe SOA. While SOA may be a predecessor to microservices, there are still differences between these architectures. Updating the methods used in SOA is possible, but creating an algorithm that can be applied to across multiple languages would be the most beneficial. Overall the most promising proposal came from Gammage et al. [2] as it detects anti-patterns in microservices at runtime.

A. DYNAMIC ANALYSIS OF SMELLS

Given the large scale of microservice architectures, dynamic analysis is necessary to get the understanding of behavior of the system at runtime. Without this insight, there is no way to observe the system in its entirety. Most papers that discussed dynamic analysis as it relates to microservices focused solely on testing the system for logic errors [17]. For the purpose of this paper, however, we are interested in applying dynamic analysis to discover anti-patterns in microservices.

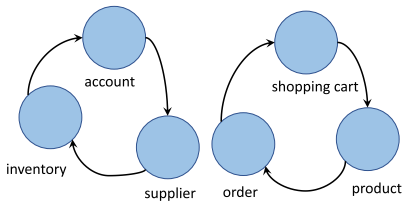


FIGURE 1. Graphical representation of cyclic dependencies.

In order to detect smells in microservices from the dynamic perspective, some papers suggested gathering information based on the log files of a system [18], [19]. Tracing through the log files and seeing how different services communicate with each other allows patterns to be found. By examining information from logs, it is possible to discover anti–patterns like cyclic dependencies because of the flow of information.

Dynamic analysis also goes hand in hand with pattern detection, as mentioned in section III. Since pattern detection involves tracking the flow of data at runtime, it is a form of dynamic analysis [20], [21]. Some papers also discuss combining static analysis with dynamic analysis in order to create the most efficient system possible [6], [20], [22]. Although these papers mention ''combining'' both forms of analysis, they are done separately and do not interact with each other. Therefore, there is not much that static analysis contributes to discovering anti–patterns that occur at runtime.

Dynamic analysis is currently not commonly used for detecting anti–patterns. Many of the papers that discuss the detection of anti–patterns utilize static analysis. More research exists on finding patterns by using dynamic analysis, and this research could be extended to include anti-patterns. Extending the existing research is likely the best solution to utilize dynamic analysis for anti-pattern detection.

B. ANTI-PATTERNS IN GRAPHS

Visualizing the architecture of a system is of great benefit to those who want a complete view of the architecture. The most basic way of visually describing a microservice-based system is through a graph [2], [23]. Having a node representing a service and an edge representing the communication between services allows for a simple understanding of how the system is working at runtime. Even with this basic visualization technique, it is possible to see anti-patterns.

Several anti-patterns can be visualized in graphs. As shown in Figure 1, cyclic dependencies are one of the simplest anti-patterns to observe. A cyclic dependency in microservice–based systems is defined as a cycle of messages that occurs between a set of services [9]. It is easy to visualize this anti-pattern because it is a closed circuit in the graph.

Bottleneck services can also be shown in graphs. A bottleneck occurs when a service is extensively used but cannot handle the high volume of requests [8]. Graphs could depict this anti–pattern by highlighting a node that has a high volume of input and output. There may be cases where developers want to have a service with a lot of communication running

through it; therefore, the maximum number of connections to other services would be determined by the developers.

The knot is another anti-pattern that can be seen in graphs. A knot is a group of services that have low cohesion, but are tightly coupled [8]. Although similar to the bottleneck, knots refer to a group of services rather than a single one. Knots can be visualized in graphs by demonstrating areas where multiple services that have a high volume of input and output are connected.

The presented anti-patterns are some of the simplest to visualize in graphs because they focus on the connections to other services. Other anti-patterns can also be visualized in graphs such as Nanoservice, Endpoint-based Service Interaction, Not Having an API Gateway, Service Chain, Shared Persistency, and Wobbly Service Interaction. Gammage et al. [2] and Borges et al. [24] describe a means for visualizing these anti-patterns.

Although anti-patterns can be visualized in graphs, this only provides a basis for visualization. The main issue regarding graphs is the fact that microservices are not just several services communicating with each other; they are many related services. Graphs are easy to observe when they are small, but modeling the architecture of a microservice in a graph can get cluttered quickly. Therefore, it is necessary to create visualization techniques that would be able to focus on the part of system that contains the anti-pattern, rather than finding anti-patterns by looking at every service.

C. VISUALIZING ANTI-PATTERNS

As mentioned in the previous section, visualizing the architecture of a system can help with the detection of anti-patterns. Seeing how the system runs and observing the flow of data offers a greater understanding of where problem areas may lie. In order to improve visualization to help understand anti-patterns in microservice-based systems, it is necessary to provide a way to automatically detect anti-patterns. While we examined the research papers for a tool that combines visualization of microservices with the detection of anti-patterns, there are few tools that achieve both. Although, there are tools that can visualize the architecture of microservices and separate tools that detect anti-patterns at runtime. Combining these tools would create a larger tool that can accomplish both visualization and detection of anti-patterns.

Visualization tools for modeling the architecture of a microservice–based system have been gaining more ground in recent years. An example visualization of a visual model of a microservice-based system is depicted in Figure 2. There are many different approaches that have been taken, but the most common way to gain a preliminary understanding of the system is to track the log files [22], [25], [26], [27]. To create a model of the system, it is necessary to discover what services are communicating with other services. One of the easiest ways to achieve this is to extract distributed tracing data[5] from log files to see what services responded

[5]https://opentracing.io

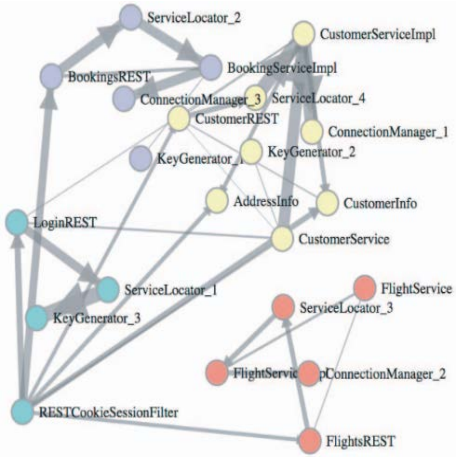


FIGURE 2. Visualization of the microservice architecture. (Reprinted from [27]).

to messages sent out by other services. This tracing forms the basis for many visualization tools. After this tracing has been finished, it varies between each tool on how the system is finally modeled.

Although visualization of microservices is achieved in several ways, there is a gap between the visualization and detection of anti-patterns. Several papers describe an approach to combine separate tools — one for visualization, one for anti–pattern detection, and another that operates between the two [24], [28], [29]. Using separate tools is currently the only way to achieve visualization of anti–patterns in microservices from the dynamic perspective. One tool that is able to complete the problem we have presented would be more beneficial because it would remove the need for multiple tools to be set up.

D. TOOLS FOR VISUALIZATION

Our research has identified a number of tools and approaches for visualizing anti–patterns through the dynamic perspective. It is important to note that visualization is distinct from detection, as detection involves reporting that a given anti–pattern exists, while visualization is the graphical representation of the anti–pattern. Table 2 indicates anti–patterns that are both detected and visualized by certain tools as well as those that are only detected and not visualized. After collecting and synthesizing the research, we found that visualization of anti–patterns was less common than tools used for their detection. Furthermore, it is important to designate the threshold for what constitutes a visualization. In order for a tool to have achieved visualization, it must include some diagram that a person could gain information from. We define tables, JSON files, and other textual descriptions to not be visualization.

After examining the presented tools, we found that the most common anti–pattern mentioned was cyclic dependencies. Many tools only obtained detection and visualization of cyclic dependencies, those being: AI Reviewer, Approach

TABLE 2. Microservice antipattern detection and visualization between tools.

Antipattern	MSANose [30]	Arcan [41]	MAIG [2]	μ Freshener [31]	Approach from Borge et al. [24]
Ambiguous Service	X				X
API Versioning					X
Bloated Service					X
Bottleneck Service			XX		X
Cyclic Dependency	X	XX	XX		
Endpoint-based Service Interaction				XX	
Enterprise Service Bus (ESB) Usage	X				
Hard-coded Endpoints	X	X			X
Inappropriate Service Intimacy	X				
Microservice Greedy	X				
Nano Service			XX		
Not Having an API Gateway	X			XX	
Service Chain			XX		
Shared Libraries	X				
Shared Persistency	X	X		XX	
The Knot			XX		
Too Many Standards	X				
Wobbly Service Interaction				XX	
Wrong Cuts	X				

Displays whether a given smell is only detected or both detected and visualized by a given tool (X = Detected; XX = Detected and Visualized).

from Mayer et al. [26], Designite [35], GSMART [22], Jaeger, Massey Architecture Explorer [36], Sonargraph [37], STAN, Structure 101, and Titan [38]. There are also two tools that only perform detection of cyclic dependencies, those tools being Hotspot Detector [39] and ARCADE [40]. Amongst the tools that only recognized cyclic dependencies, most performed the detection automatically, but the approach from Mayer et al. [26] and Jaeger required manual detection. There are different approaches to visualization and they are listed in Table 3, but cyclic dependencies were most commonly shown in service dependency graphs (SDG). Although there are many tools that can only detect cyclic dependencies, there is room for the extension of these tools. Extending the visualization and detection techniques of these tools to include more anti–patterns would offer new approaches to the removal of anti–patterns.

Although many tools we found only focused on cyclic dependencies, there are several tools listed in Table 2 that did include other anti–patterns. MSANose [30] is one of the tools that mentioned the detection of other anti–patterns. Although there is mention of other anti–patterns, MSANose detects anti–patterns using static analysis and does not provide any visualization. Although using static analysis to detect anti–patterns is a valid approach, our mapping study focuses on using dynamic analysis. MSANose is mentioned here, however, because of its ability to detect other anti–patterns not included by other tools.

Arcan is another tool we found that is able to detect other anti–patterns. Pigazzini et al. [41] extended Arcan in order to achieve the detection of three anti–patterns, as well as the visualization of cyclic dependencies. This is a smaller number of detected anti–patterns than MSANose, and it also achieves detection by static analysis. Although Arcan's extension is able to detect more than cyclic dependencies, MSANose achieves more with static analysis.

Borges et al. [24] present another means for detecting several anti–patterns. The approach is built off of Spinnaker and detects a total of five anti–patterns. This approach still finds fewer anti–patterns than MSANose and also utilizes static analysis. Although it may find fewer anti–patterns, it can detect three anti–patterns that MSANose does not. However, because it is focused on static analysis, this tool does not provide a means for finding anti–patterns at runtime.

Microservice Anti–Patterns Insights Generator (MAIG) [2] is capable of both visualizing and detecting anti–patterns based on data obtained at runtime. MAIG operates by using dependency graphs to provide a view of how the system is behaving. There are also tracking tools that follow the number of outgoing and incoming edges for a service, which aid in the detection of anti-patterns such as the knot and bottleneck. Operating based on trace data obtained from Zipkin and relaying that data to the graph database Neo4j, MAIG successfully combines two different tools to achieve visualization and detection of anti–patterns.

Lastly, μFreshener [31] is also able to detect and visualize anti–patterns based on information obtained at runtime. By processing collected trace information, μFreshener is able to detect anti–patterns and visualize them by highlighting the impacted nodes in the corresponding service graph. μFreshener can also be combined with other tools like μTOSCA in order to achieve this detection.

After analyzing all of the tools, it is apparent that there are not many existing tools for detecting and visualizing anti–patterns. The two that achieved this, MAIG and μFreshener, also relied on other tools to achieve the visualization and detection. Combining tools is therefore the only approach we found to properly visualize anti–patterns in microservices from the dynamic. No single tool is completely capable of providing all of the aspects our study intended to find.

TABLE 3. Methods of visualizations for each microservice antipattern.

Antipattern	Visualization Methods
Bottleneck Service	Dependency Graph [2]
Cyclic Dependency	SDG [22], [26], [36], [37]; Sunburst Diagram [35]; Design Structure Matrix [38]
Endpoint-based Service Interaction	SDG [31]
Nano Service	Dependency Graph [2]
Not Having an API Gateway	SDG [31]
Service Chain	Dependency Graph [2]
Shared Persistency	SDG [31]
The Knot	Dependency Graph [2]
Wobbly Service Interaction	SDG [31]

E. FUTURE DIRECTIONS

Many papers identified in this mapping study mentioned some future work in their conclusion. We found that there were three general ideas mentioned across most of the papers we collected. First, papers that focused on pattern detection or definitions of anti–patterns wanted to extend their understanding to include more patterns [14], [19] as well as improve their pattern detection technique [11], [25]. Improving pattern detection to include more patterns may lead to the addition of anti–patterns, thus creating a new means for anti–pattern detection. For defining new anti–patterns, the more common issues are found across microservices, the more solutions will arise that can be applied across all microservices. Improving pattern detection will assert that the given tool will perform more accurately, allowing less anti–patterns to avoid being found.

Second, papers that focused on visualizing the architecture of a system discussed improving the adaptability of their visualization tool in the future [4], [10], [18]. Providing a way for a visualization tool to grow as a microservice-based system evolves would prove to be a major asset. As mentioned by Nakazawa et al. [27], there is a limit on the maximum number of nodes that can exist in their visualization tool, which would lead to a problem whenever a system grew too large for that tool. Another future direction related to visualization that was only mentioned by Zhao et al. [21] was the idea to take the visual model of a system and create a coding framework based on the model. This could allow for the design of a microservice–based system from a visual perspective, such that anti–patterns are not included from the start of the process.

Finally, the last common future direction mentioned was an automatic tool for testing the quality of a system [3], [7], [17], [26]. While the quality of a system typically refers to the system working as intended, the papers mentioned extending these tools to find design issues. Design issues can be related to anti–patterns, and therefore have some interest in our mapping study. Another future direction mentioned in Sampaio et al. [28] discusses improving log analysis tools to increase the accuracy and the amount of tracked data.

F. THREATS TO VALIDITY

The main threat to validity comes from the exclusion of papers that could have been relevant to visualizing anti–patterns in microservices from the dynamic perspective.

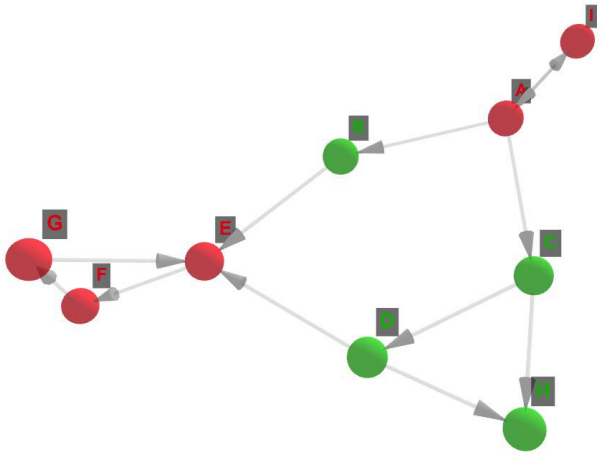


FIGURE 3. Our tool prototype to visualize anti–patterns in a microservice system service dependency graph.

This exclusion comes in phase 2 of section III from our search query. Since this search is performed automatically, we confirmed whether a query gave us valid results based only on the total number of results and the first 25 papers listed. Relevant papers could have been excluded here if they did not contain keywords that were listed in our query, thus leading to the potential exclusion of tools. In order to mitigate this issue, we made our query broad in order to avoid losing papers that were relevant to the topic. We also expanded our query to include service-oriented architecture to avoid missing any written research on the predecessor of microservices.

Another threat comes from the potential exclusion of papers in the third phase — the filtering of the titles and abstracts. Human error could have caused some papers that may have had relevance to be lost, but to avoid any major concern with this issue, multiple authors read through each title and abstract. Having multiple authors in this phase reduced the chance that a relevant paper would accidentally be removed because each title and abstract was read at least twice, by two different people. When there were disagreements between the authors, another author was brought in to resolve the inconsistency.

The last threat to validity comes from the number of search indexers used. Only five were selected for this paper, namely ACM Digital Library (DL), IEEE Xplore, ScienceDirect, Scopus, and SpringerLink. There may be other papers that

were published on other sites that are not included in our study. To mitigate this issue, the related works were read in the relevant papers in order to include any research that may have been published on a different site.

V. CONCLUSION

Visualizing anti–patterns in microservices from the dynamic perspective helps their detection and removal. Removing anti–patterns from a system's design helps to improve the quality of service as well as prevent potential design flaws that could appear as the system evolves. Our mapping study found a total of 31 papers relevant to our topic and after thoroughly examining them, we presented the current state of research on our topic. We found that there is research discussing the detection of anti–patterns at runtime, and there is separate research visualizing a microservice-based system from the dynamic perspective. Combining these methods results in an overall system that can detect anti–patterns and notify a visualization tool of their existence. While there was not an individual tool that was able to accomplish both aspects of our topic, this combination of tools is the most common approach we found.

For future research, we are developing our own tool that is able to detect selected anti–patterns at the service-dependency graph and uses visualization to highlight the occurrence of the anti–pattern. Figure 3 shows our tool visualization demonstrating cyclic dependencies. Our study helped us to gain a stronger understanding of the current outlook on anti-patterns in microservices and how they are visualized from the dynamic perspective, as well as what research may appear in the future.

REFERENCES

[1] J. Rasheedh and S. Saradha, "Design and development of resilient microservices architecture for cloud based applications using hybrid design patterns," *Indian J. Comput. Sci. Eng.*, vol. 13, no. 2, pp. 365–378, 2022.

[2] I. U. P. Gamage and I. Perera, "Using dependency graph and graph theory concepts to identify anti-patterns in a microservices system: A tool-based approach," in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, Jul. 2021, pp. 699–704.

[3] F. Z. Safy, M. El-Ramly, and A. Salah, "Runtime monitoring of SOA applications: Importance, implementations and challenges," in *Proc. IEEE 7th Int. Symp. Service-Oriented Syst. Eng.*, Mar. 2013, pp. 315–319.

[4] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kroger, "Microservice decomposition via static and dynamic analysis of the monolith," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2020, pp. 9–16.

[5] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, and L. Su, "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1387–1397, doi: 10.1145/3368089.3417066.

[6] A. Ouni, M. Kessentini, K. Inoue, and M. O. Cinnéide, "Search-based web service antipatterns detection," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 603–617, Aug. 2017.

[7] S. Panichella, M. Rahman, and D. Taibi, "Structural coupling for microservices," in *Proc. 11th Int. Conf. Cloud Comput. Services Sci. (CLOSER)*. SCITEPRESS—Science and Technology Publications, pp. 280–287, doi: 10.5220/0010481902800287.

[8] F. Palma and N. Mohay, "A study on the taxonomy of service antipatterns," in *Proc. IEEE 2nd Int. Workshop Patterns Promotion Anti-Patterns Prevention (PPAP)*, Mar. 2015, pp. 5–8.

[9] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE Softw.*, vol. 35, no. 3, pp. 56–62, May/Jun. 2018.

[10] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle, "Towards recovering the software architecture of microservice-based systems," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 46–53.

[11] J. P. Sotomayor, S. C. Allala, D. Santiago, T. M. King, and P. J. Clarke, "Comparison of open-source runtime testing tools for microservices," *Softw. Quality J.*, May 2022.

[12] M. Di Penta, A. Santone, and M. L. Villani, "Discovery of SOA patterns via model checking," in *Proc. 2nd Int. Workshop Service Oriented Softw. Eng. Conjunct 6th ESEC/FSE Joint Meeting*. Association for Computing Machinery, Sep. 2007, pp. 8–14, doi: 10.1145/1294928.1294931.

[13] K. Vidackovic and A. Weisbecker, "A methodology for dynamic service compositions based on an event-driven approach," in *Proc. Annu. SRII Global Conf.*, Mar. 2011, pp. 484–494.

[14] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0," *Knowl.-Based Syst.*, vol. 89, pp. 97–112, Nov. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705115002397

[15] T. Holmes, E. Mulo, U. Zdun, and S. Dustdar, "Model-aware monitoring of SOAs for compliance," in *Service Engineering: European Research Results*, pp. 117–136.

[16] P. Scandurra and S. Capelli, "A practical and automated approach for engineering service-oriented applications with design patterns," in *Proc. IEEE 38th Int. Comput. Softw. Appl. Conf. Workshops*, Jul. 2014, pp. 684–689.

[17] D. Rud, A. Schmietendorf, and R. Dumke, "R.: Product metrics for service-oriented infrastructures," in *Proc. Appl. Softw. Meas. Int. Workshop Software Metrics DASMA Softw. Metrik Kongress (IWSM/MetriKon*, Jan. 2006.

[18] F. H. Vera-Rivera, E. Puerto, H. Astudillo, and C. M. Gaona, "Microservices backlog—A genetic programming technique for identification and evaluation of microservices from user stories," *IEEE Access*, vol. 9, pp. 117178–117203, 2021.

[19] M. Nayrolles, N. Moha, and P. Valtchev, "Improving SOA antipatterns detection in service based systems by mining execution traces," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 321–330.

[20] F. Palma, N. Moha, G. Tremblay, and Y.-G. Guéhéneuc, "Specification and detection of SOA antipatterns in web services," in *Software Architecture* (Lecture Notes in Computer Science), P. Avgeriou and U. Zdun, Eds. Berlin, Germany: Springer, 2014, pp. 58–73.

[21] B. Zhao, Y. Zhao, and D. Ma, "A constraint mechanism for dynamic evolution of service oriented systems," in *Proc. IEEE 15th Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, Apr. 2012, pp. 103–110.

[22] S.-P. Ma, C.-Y. Fan, Y. Chuang, I.-H. Liu, and C.-W. Lan, "Graph-based and scenario-driven microservice analysis, retrieval, and testing," *Future Gener. Comput. Syst.*, vol. 100, pp. 724–735, Nov. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19302614

[23] Y. Zuo, X. Zhu, J. Qin, and W. Yao, "Temporal relations extraction and analysis of log events for micro-service framework," in *Proc. 40th Chin. Control Conf. (CCC)*, Jul. 2021, pp. 3391–3396.

[24] R. Borges and T. Khan. *Algorithm for Detecting Antipatterns in Microservices Projects*. CEUR-WS. Accepted: Mar. 30, 2021. [Online]. Available: https://trepo.tuni.fi/handle/10024/129802

[25] L. Mazzola, P. Kapahnke, and M. Klusch, "Semantic composition of optimal process service plans in manufacturing with ODERU," *Int. J. Web Inf. Syst.*, vol. 14, no. 4, pp. 495–523, 2018.

[26] B. Mayer and R. Weinreich, "An approach to extract the architecture of microservice-based software systems," in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Mar. 2018, pp. 21–30.

[27] R. Nakazawa, T. Ueda, M. Enoki, and H. Horii, "Visualization tool for designing microservices with the monolith-first approach," in *Proc. IEEE Work. Conf. Softw. Visualizat. (VISSOFT)*, Sep. 2018, pp. 32–42.

[28] A. R. Sampaio, H. Kadiyala, B. Hu, J. Steinbacher, T. Erwin, N. Rosa, I. Beschastnikh, and J. Rubin, "Supporting microservice evolution," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 539–543.

[29] H. Fernández, C. Tedeschi, and T. Priol, "Decentralized workflow coordination through molecular composition," in *Proc. Int. Conf. Service-Oriented Comput.*, in Lecture Notes in Computer Science, vol. 7221, 2011, pp. 22–32.

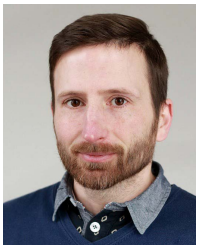
[30] A. Walker, D. Das, and T. Cerny, "Automated code-smell detection in microservices through static analysis: A case study," *Appl. Sci.*, vol. 10, no. 21, p. 7800, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/21/7800
[31] J. Soldani, G. Muntoni, D. Neri, and A. Brogi, "The μ TOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures," *Softw., Pract. Exper.*, vol. 51, no. 7, pp. 1591–1621, Jul. 2021.
[32] F. Ponce, J. Soldani, H. Astudillo, and A. Brogi, "Smells and refactorings for microservices security: A multivocal literature review," *J. Syst. Softw.*, vol. 192, Oct. 2022, Art. no. 111393. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122200111X
[33] V. Bushong, A. S. Abdelfattah, A. A. Maruf, D. Das, A. Lehman, E. Jaroszewski, M. Coffey, T. Cerny, K. Frajtak, P. Tisnovsky, and M. Bures, "On microservice analysis and architecture evolution: A systematic mapping study," *Appl. Sci.*, vol. 11, no. 17, p. 7856, Aug. 2021.
[34] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, Aug. 2015.
[35] T. Sharma. (May 2016). *Designite—A Software Design Quality Assessment Tool*. [Online]. Available: http://www.designite-tools.com
[36] J. Dietrich, "Upload your program, share your model," in *Proc. Annu. Conf. Syst., Program., Appl., Softw. Humanity*, 2012, pp. 21–22.
[37] A. von Zitzewitz, "Mitigating technical and architectural debt with sonargraph," in *Proc. IEEE/ACM Int. Conf. Tech. Debt (TechDebt)*, May 2019, pp. 66–67.
[38] L. Xiao, Y. Cai, and R. Kazman, "Titan: A toolset that connects software architecture with quality analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2014, pp. 763–766.
[39] R. Mo, Y. Cai, R. Kazman, and L. Xiao, "Hotspot patterns: The formal definition and automatic detection of architecture smells," in *Proc. 12th Work. IEEE/IFIP Conf. Softw. Archit.*, May 2015, pp. 51–60.
[40] D. M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "An empirical study of architectural change in open-source software systems," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, May 2015, pp. 235–245.
[41] I. Pigazzini, F. A. Fontana, V. Lenarduzzi, and D. Taibi, "Towards microservice smells detection," in *Proc. 3rd Int. Conf. Tech. Debt*, Jun. 2020, pp. 92–97.

GARRETT PARKER is currently a Junior Student in computer science with Baylor University. Since Fall 2020, he has been featured on the Baylor University Dean's List. His research interests include static and dynamic code analysis and visualization. In addition, he received the Computer Science Scholarship Award, in Spring 2022.

SAMUEL KIM is currently pursuing the bachelor's degree in computer science with Baylor University. He has been recognized on the Baylor University Dean's List and with scholarship awards, including the Baylor Computer Science Scholarship and the Baylor Association of Computing Machinery Scholarship. His research interests include the visualization of distributed systems and the security of computer systems.

ABDULLAH AL MARUF (Member, IEEE) received the bachelor's degree from the Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Bangladesh. He is currently pursuing the master's degree in computer science with Baylor University. He has four years of professional experience as a Software Developer and a DevOps Engineer. He is an Open-Source Enthusiast. His research interests include software engineering, code analysis, and runtime log analysis.

TOMAS CERNY received the master's and Ph.D. degrees from the Faculty of Electrical Engineering (FEE), Czech Technical University in Prague, and the M.S. degree from Baylor University. In 2009, he started his academic career at the FEE, Czech Technical University, from where he transferred to Baylor University, in 2017. He is currently a Professor of computer science with Baylor University. He worked more than ten years as the Lead Developer for the International Collegiate Programming Contest Management System. He authored over 100 publications, mostly related to code analysis and enterprise systems. His research interests include software engineering, cloud systems, and code analysis. Among his awards are best papers at Microservices 2022, IEEE SOSE 2022, Closer 2022, LXNLP 2022, the Outstanding Service Award ACM SIGAPP 2018 and 2015, and the 2011 ICPC Joseph S. DeBlasi Outstanding Contribution Award. He served on the committee of multiple conferences in the past few years, including Program Chair or Conference Chair at ACM SAC, ACM RACS, and ICITCS.



KAREL FRAJTAK received the master's and Ph.D. degrees from the Faculty of Electrical Engineering, Czech Technical University in Prague. He is currently a Lecturer and a Researcher with the System Testing Intelligent Laboratory (STILL), Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague.

PAVEL TISNOVSKY received the Ph.D. degree from the Brno University of Technology, Czech Republic. He was an Assistant Professor, from 1999 to 2005. He is currently a Principal Quality Engineer with Red Hat Inc., with over ten years of experience. He is a programming language enthusiast and the author of many articles and series at Linux magazine ROOT.cz. He holds one software patent on testing and also works on tools for OpenShift.io—open development services for creating, building, and testing container applications.

DAVIDE TAIBI (Member, IEEE) has been a member of the International Software Engineering Network (ISERN), since 2018. He is currently a Full Professor with the University of Oulu, Finland, where he is also the Head of the M3S Cloud Research Group. Before moving to Finland, he was an Assistant Professor at the Free University of Bozen/Bolzano, from 2015 to 2017; a Postdoctoral Research Fellow at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering—IESE, from 2013 to 2014; and a Research Fellow at the University of Insubria, from 2007 to 2011. His research interests include empirical software engineering applied to cloud-native systems, with a special focus on the migration from monolithic to cloud-native applications. He is investigating processes and techniques for developing cloud native applications, identifying cloud-native specific patterns and anti-patterns.