Understanding Data Access in Microservices Applications Using Interactive Treemaps

Maxime André*, Marco Raglianti†, Anthony Cleve*, Michele Lanza†
*Namur Digital Institute, University of Namur, Belgium
†REVEAL @ Software Institute — USI, Lugano, Switzerland

***Abstract*—Over the past decade, microservices have gained significant popularity, impacting how applications are designed and deployed. Maintaining a comprehensive high-level view of microservices applications is essential, especially for software evolution tasks, enabling developers to understand, maintain, and optimize the complex interactions across various services.**

Developers struggle to obtain such an overview, particularly from a data perspective. Currently, when changes occur, they must identify data access code fragments dependent on the modified parts, or manually search through the entire codebase for potentially impacted ones. This process is time-consuming, error-prone, and cumbersome, especially in large codebases residing in multiple repositories and accessing multiple databases.

We present a novel approach to support code and data co-evolution comprehension. We mine data access fragments using a custom static analyzer and use interactive treemaps to generate a high-level view of the architecture, which can be explored at various levels of detail allowing, among the others, several and quick what-if analyses to assess the impact of changes (*e.g.,* data concept modification, technology switch).

As a case study, we use Overleaf, a popular online LaTeX collaborative authoring platform, to evaluate our approach. We compared multiple versions and analyzed the evolution of 1.9k code fragments associated to more than 350 data concepts across 13 microservices, 855 directories, and 3.5k files mixing different data access technologies. We complement our analysis with insights and reflections on the promising approach.

***Index Terms*—microservices, data access, treemap, program comprehension**

I. Introduction

Microservices as a software architecture pattern were first "officially" defined in 2011 [1]. Since then, they have grown in popularity and are now widely adopted by many software companies, including *Amazon*, *Google*, and *Netflix* [2], [3]. They are particularly praised for their *modularity*, *heterogeneity*, and *interoperability* capabilities [4]. In theory, these qualities are supposed to ease software evolution. However, recent studies report that the same capabilities affect the difficulty in obtaining an overview of the system [5]–[8].

From a data management perspective, microservices architectures constitute an additional layer of complexity. The *polyglot persistence* [1], incarnated by multiple, heterogeneous, and distributed databases, impacts system overviews [9]. When changes occur, developers must *know* data access code fragments dependent on the modified parts, or manually *search* through the entire codebase to identify potentially impacted ones. This process is time-consuming, error-prone, and cumbersome [10], especially in large codebases residing in several repositories and accessing multiple databases.

Several studies have explored the holistic visualizations of microservices architectures [5], [7], [8], [11]. These works typically focus on interactions between microservices from a software architecture point of view, neglecting a deeper analysis of the data access layer. Since most existing approaches rely on dynamic analysis, they require the deployment of the system to perform the proposed analyses and generate their insights. Finally, software visualizations for microservices predominantly rely on directed graphs layouts [5], [7], [11], which can limit interpretability for large systems [8]. Developers need a comprehensive and scalable high-level overview of microservices applications that incorporates the data access layer as a "first-class citizen", to help them understand, maintain, and optimize the complex interactions across multiple microservices during software evolution tasks.

We propose a novel approach to support the comprehension of code and data co-evolution in microservices applications. We mine data access fragments and present them in a treemap-based visualization. The treemap layout allows a scalable and compact high-level overview of the system, still providing precise localizability of the data access fragments in the codebase. The visualization is interactive, enabling developers to explore the system at different levels of detail and progressively disclose information such as fragments accessing the same data concepts (*i.e.,* database conceptual entities).

Our interactive treemap supports several *what-if* analyses, helping developers assess the impact of changes (*e.g.,* data concept modifications, technology switches). It also offers insights into the distribution of technologies, operations, code fragments, and data concepts, highlighting heterogeneity or consistency at macro (*i.e.,* entire codebase) and micro scales (*i.e.,* in a single file). Our approach assists in evaluating the importance of each microservice and component from a data access perspective and helps identifying the data-oriented parts of a system and their characteristics. Finally, in terms of quality assessment, it contributes to finding anti-patterns for refactorings, such as isolated or highly nested code fragments.

In the following sections, we present our approach (Section II) and a case study (Section III), with reflections on some of the what-if scenarios we envision as most important to support developers in the source code and data co-evolution comprehension of microservices architectures. Section IV positions our work within the existing literature, while Section V presents future research directions.

 DOI: 10.1109/ICPC66645.2025.00030

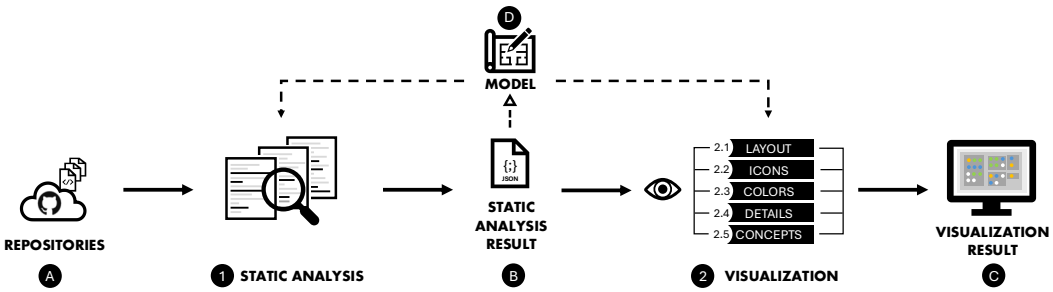

Fig. 1. Approach overview.

II. Interactive Visualization of Data Access

Interactive visualization is the key to exploration and comprehension of code and data co-evolution. An overview of our approach is presented in Figure 1. From source code repositories (A) we use a static analyze (1) [12], [13] to retrieve code fragments (B) identifying data access in the code. We generate an interactive treemap (2) to present an explorable and customizable high-level architectural view of the system (C). The underlying model (D) allows exploration of the elements in the visualization and traceability to the original software repository. For example, one can open the source code file where each code fragment is located to further inspect the fragment itself and its context of use.

A. Static Analysis

We mine the microservices architecture of interest following the approach of *André et al.* [12]. We download one or more Git repositories (A). We parse and query the source code to identify code fragments that represent data access. We extract the relevant information to produce an analysis report as output (B). This report represents a microservices architecture as a set of repositories, which are subdivided into directories and files containing collections of code fragments. Each code fragment is enhanced with additional details, such as the database technology used, the CRUD operation (Create, Read, Update, or Delete), the specific Object-Relational Mapping (ORM) method employed, and, when available, a sample of the data object or value affected by the operation. Each code fragment may be linked to one or more data concepts. Those concepts are extracted by applying a few Natural Language Processing rules (*e.g.,* lemmatization) to the fragment data sample. Our analysis report relies on a custom underlying model (D).

B. Visualization

We use the analysis report as input to create the visualization. We recursively traverse the repository's hierarchy in the report, building different aspects of the visualization (*e.g.,* layout, icons, colors, as presented in Figure 1, 2.1–2.5) before generating the final result, the interactive treemap (C).

Figure 2 shows how the containment hierarchy of repositories, directories, and files contributes directly to the treemap layout. Literature demonstrates that treemaps are well-suited for visualizing hierarchical and recursive large structures in limited 2D spaces [14]–[21], which aligns with our model.

We apply a bin-packing *First Fit Decreasing* heuristic algorithm[1] to pack the rectangles representing files and directories. Since each code fragment has a defined and uniform size, the layout is constructed bottom-up. The size of each parent rectangle depends on the number and sizes of its children.

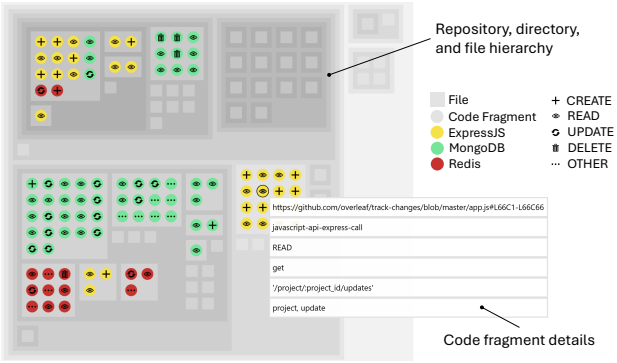

Fig. 2. Visualization result example.

Figure 2 also shows that the code fragment's *operation* in the model determines the icon used to visually represent the fragment. Each *technology* is mapped to a different configurable color. Additional details are integrated in the visualization to help interactive navigation. For example, when hovering over a box, a tooltip appears with information such as the hovered file path or the code fragment location. Finally, we use the list of related data concepts to interactively highlight the code fragments related to selected concepts. These multiple layers of information contribute to the codebase exploration and support several comprehension and evolution tasks.

III. Case Study

Our case study concerns Overleaf, a popular collaborative LaTeX authoring platform. We compared several versions, the latest of which (Figure 3, right) contains up to 1.9k code fragments associated to more than 350 data concepts across 13 microservices, 855 directories, and 3.5k files mixing different data access technologies. In total, this codebase consists of 800k LoC. Overleaf is open source and its GitHub repository[2] reaches 14k stars and 103 contributors. Below we detail 4 main observations we made thanks to our visualization.

[1]See `https://www.npmjs.com/package/bin-pack`
[2]See `https://github.com/overleaf/overleaf`

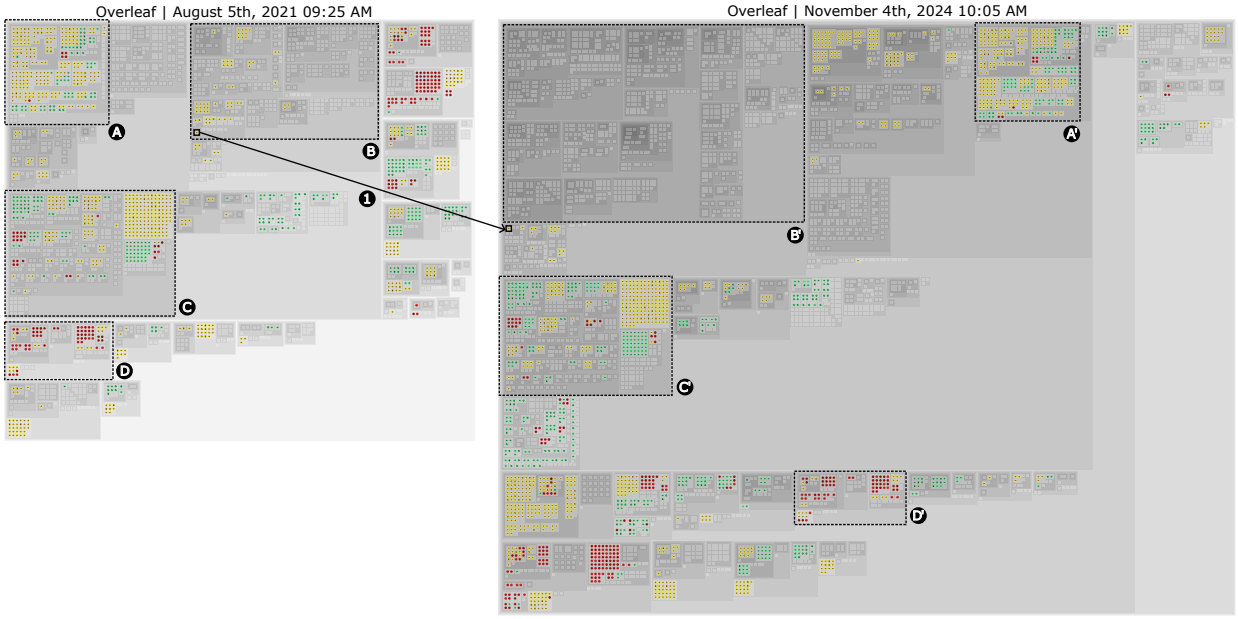

Fig. 3. Comparison of two versions of Overleaf.

A. Technology Breakdown

The treemap helps us to understand at a glance the technologies used in the system. As depicted in Figure 4, we identify two scenarios: Either the technologies are mixed within the same files (1), or they are well-distributed and encapsulated (2). This gives us an indication of how the data access code is organized and how heterogeneous it is. The observation presented at micro level (single files) is also valid at macro level (microservices and the entire system, see Figure 3).

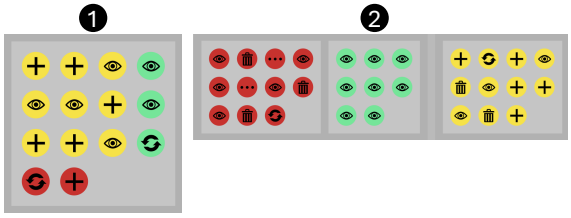

Fig. 4. Single file with mixed technologies (left) and multiple files each with a single technology (right).

B. Data Concept Change Impact

The interactive visualization allows highlighting the code fragments related to specific data concepts. This is useful to assess the impact of changing a data concept on the affected fragments and their distribution in the codebase. As depicted in Figure 5, code fragments that are not related to the selected data concept are rendered opaque. In this example, we can see how CRUD operations on the *"doc"* concept are spread across multiple components of the Overleaf codebase (1), while the *"tag"* concept is cohesively independent (2). In addition, we can quickly determine which technologies manipulate a given concept, supporting decisions for switching technologies.

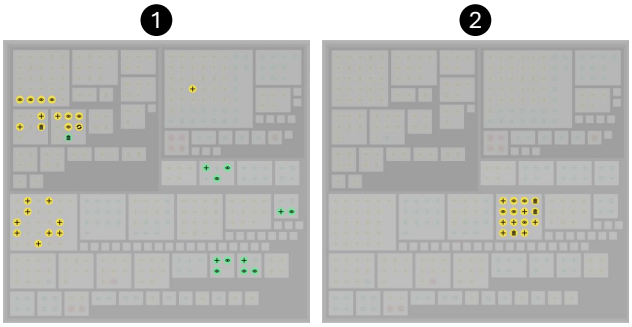

Fig. 5. Example of highlighted data concepts. CRUD operations with ExpressJS and MongoDB on the *"doc"* concept (left) are spread in 9 files. Operations on the *"tag"* concept are single technology and single file (right).

C. Version Comparison

The previous observations focused on a single version of the codebase. We can also compare two versions of the same project. Figure 3 shows the result of three years of evolution of Overleaf. While it is apparent that the system grew, we observe that some parts of the system remained stable. At a first glance, we can even see identical blocks in both versions (A vs. A', C vs. C', and D vs. D'). The compared visualizations also suggest that some parts have been refactored (B). For instance, data access code fragments have been removed from the front-end (B') to increase the separation of concerns.

D. Code Fragment Isolation

Besides highlighting the result of refactorings, comparisons on Overleaf reveal another use case. Some code fragments are particularly isolated, highly nested in the codebase. Although this can result from a motivated choice, they deserve specific attention as a potential anti-pattern [22].

Figure 3 shows an instance of isolated code fragment in the older version of Overleaf (Aug. 5, 2021). The recent version (Nov. 4, 2024) shows that the code fragment was moved to another location (1). This move seems to relate to the `/frontend/js/` refactoring described above. Overleaf's developers successfully located and adapted this code fragment, yet this type of instance still remain difficult to locate. Our visualization helps to identify them faster than via manual search.

IV. Related Work

Cerny et al. review techniques that provide holistic views of microservices architectures. The authors aim to fill the lack of a reified view caused by decentralized architectures, especially for large systems involving several independent teams. In a first work [8], besides the most commonly used visualizations based on directed graphs, the authors discuss alternative kinds of layout, such as the notation-based ones, leveraging UML and SysML models. They also mention the matrix-based approach and the metaphor-based visualization using physical world contexts such as cities, islands, or landscapes. They point-out that 3D spaces in *Augmented Reality* and *Virtual Reality* can contribute to the understandability. In addition, they present tools such as Amazon's *X-Ray console*[3], *Simianviz*[4], *MicroDepGraph*[5]. In another review [5], they list tools like *VR-EA* [23], *Kiali*[6], and *Jaeger UI*[7]. They also approach map-based and hierarchical visualizations. The same authors introduce Microvision [24], a 3D-based visualization tool able to visualize holistically a microservices architecture with complex relationships in *Augmented Reality*. They point-out that the 3D allows exploiting a larger area, thus suitable for large codebases, and that the navigation helps to inspect, in a more natural way, various angles and abstraction perspectives.

From a 3D perspective, *Abdelfattah et al.* investigate *Virtual Reality* for visualizing microservices architectures, especially for representing inter-services dependencies. Based on a controlled experiment, they observe that the third dimension helps developers to identify the dependencies, the cardinalities, and the potential bottlenecks [25].

Ardigò et al. present *M3tricity*, a tool that uses the city metaphor also in 3D to visualize the evolution of object-oriented software systems, with particular emphasis on the data and information access. According to the authors, the tool contributes to the program comprehension and the evolution analysis [26], [27].

Parker et al. conduct a systematic mapping study on the visualization of microservices anti-patterns at runtime. They argue that bringing a way to observe large microservices architectures as a whole would offer a great benefit to microservices developers. Similarly to *Cerny et al.*, they find a prevalence of visualizations based on directed graphs [11].

Ma et al. introduce an approach for creating graphs that visualize microservice dependencies for analysis and testing. It helps detect issues early and track service connections during updates, working well for systems of any size [28].

In a systematic mapping study, *Gortney et al.* review dynamic analysis techniques for visualizing microservices architectures. They emphasize that the absence of views of the entire system could lead to a messy architecture and potential problems in terms of change propagation. Directed graph-based visualization is the most common approach and it has been used in different variants such as the service dependency graph, the business process graph, and the service endpoint call graph. Additionally, the study reveals that some tools rely on 3D-based *Augmented Reality*. The authors cite tools like *MicroART* [29], *Zipkin*[8], and *ExplorViz*[9] [7].

Mayer et al. propose a dashboard for visualizing dependency graphs and multiple charts of metrics taken from the information extracted statically and dynamically from microservices architectures [30].

The common thread in the related work is that each approach tries to tame the complexity of the domain, where the intrinsic complexity of software is augmented with the additional complexity of data storage mechanisms. Starting from a simplification approach, our idea follows Ben Shneiderman's mantra: *"Overview first, zoom and filter, then details-on-demand"* [31]. We use a simple yet sufficient 2D representation (treemaps), making it interactive, and augmenting it with carefully selected meta-information about data access.

V. Conclusions

We addressed the challenge of maintaining a high-level overview of microservices architectures and their data access by introducing a novel treemap-based visualization approach, offering a fresh alternative to traditional graph-oriented representations. Our interactive treemap enables us to perform what-if analyses, assess the distribution of technologies and data concepts in the codebase, and evaluate the size and heterogeneity of data access components.

Through a case study on Overleaf, we demonstrated the performance of the proposed approach in identifying technology breakdown patterns, analyzing the impact of data concept changes, comparing system versions to track evolution or refactoring scenarios, and highlighting particular code patterns.

This promising approach can be used in future work to aid in decision-making for refactoring and evolution strategies. It also opens the door to extend the visualization with additional dimensions for even richer insights.

VI. Replication Package

To ensure verifiability and replicability of our work, the artifacts of our case study are publicly available as open source at: https://figshare.com/s/4dce410c10c7c26dabe3

[3]See **https://aws.amazon.com/xray/**
[4]See **https://simianviz.surge.sh/**
[5]See **https://github.com/clowee/MicroDepGraph**
[6]See **https://kiali.io/**
[7]See **https://github.com/jaegertracing/jaeger-ui**
[8]See **https://zipkin.io/**
[9]See **https://github.com/ExplorViz**

ACKNOWLEDGMENTS

This research is supported by the SofinaBoël Fund for Education and Talent and the Federation Wallonie-Bruxelles (FWB), as part of the ARC project RAINDROP, and by the Swiss National Science Foundation (SNSF) through the project "FORCE" (SNF Project No. 232141). The authors would like to thank the Swiss Group for Original and Outside-the-box Software Engineering (CHOOSE) for sponsoring the trip to the conference.

REFERENCES

[1] J. Lewis and M. Fowler. (2014) Microservices. [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] C. Richardson, *Microservices Patterns: with Examples in Java*. Simon and Schuster, 2018.

[3] S. Newman, *Building microservices*. O'Reilly Media, Inc., 2021.

[4] M. André, "Automated database schema evolution in microservices," in *Conference on Very Large Data Bases (VLDB)*, vol. 3452. CEUR-WS, 2023, pp. 37–40.

[5] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice architecture reconstruction and visualization techniques: A review," in *IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 39–48.

[6] T. Cerny and D. Taibi, "Static analysis tools in the era of cloud-native systems," in *International Conference on Microservices (Microservices)*, 2022, arXiv preprint. [Online]. Available: https://arxiv.org/abs/2205.08527

[7] M. E. Gortney, P. E. Harris, T. Cerny, A. Al Maruf, M. Bures, D. Taibi, and P. Tisnovsky, "Visualizing microservice architecture in the dynamic perspective: A systematic mapping study," *IEEE Access*, vol. 10, pp. 119 999–120 012, 2022.

[8] T. Cerny, A. S. Abdelfattah, J. Yero, and D. Taibi, "From static code analysis to visual models of microservice architecture," *Cluster Computing*, pp. 1–26, 2024.

[9] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, "Data management in microservices: State of the practice, challenges, and research directions," *VLDB Endowment*, vol. 14, no. 13, pp. 3348–3361, 2021.

[10] A. Lercher, J. Glock, C. Macho, and M. Pinzger, "Microservice API evolution in practice: A study on strategies and challenges," *Journal of Systems and Software*, vol. 215, p. 112110, 2024.

[11] G. Parker, S. Kim, A. Al Maruf, T. Cerny, K. Frajtak, P. Tisnovsky, and D. Taibi, "Visualizing anti-patterns in microservices at runtime: A systematic mapping study," *IEEE Access*, vol. 11, pp. 4434–4442, 2023.

[12] M. André, E. Rivière, and A. Cleve, "Data access-centered understanding of microservices architectures," in *IEEE International Conference on Software Architecture (ICSA): Companion Proceedings*. IEEE, 2025, in press.

[13] M. André, E. Rivière, and A. Cleve. DENIM reverse engineering. Zenodo. [Online]. Available: https://doi.org/10.5281/zenodo.14740539

[14] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Transactions on Graphics*, vol. 11, no. 1, pp. 92–99, 1992.

[15] M. Balzer and O. Deussen, "Exploring relations within software systems using treemap enhanced hierarchical graphs," in *IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE, 2005, pp. 1–6.

[16] M. Balzer, O. Deussen, and C. Lewerentz, "Voronoi treemaps for the visualization of software metrics," in *ACM Symposium on Software Visualization (SoftVis)*. ACM, 2005, pp. 165–172.

[17] R. V. Hees and J. Hage, "Stable and predictable Voronoi treemaps for software quality monitoring," *Information and Software Technology*, vol. 87, pp. 242–258, 2017.

[18] E. Faccin Vernier, A. C. Telea, and J. Comba, "Quantitative comparison of dynamic treemaps for software evolution visualization," in *IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2018, pp. 96–106.

[19] W. Scheibel, C. Weyand, and J. Döllner, "EvoCells—A treemap layout algorithm for evolving tree data," in *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, vol. 3: IVAPP, 2018, pp. 273–280.

[20] W. Scheibel, M. Trapp, D. Limberger, and J. Döllner, "A taxonomy of treemap visualization techniques," in *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, vol. 3: IVAPP, 2020, pp. 273–280.

[21] D. P. Tua, R. Minelli, and M. Lanza, "Voronoi evolving treemaps," in *Working Conference on Software Visualization (VISSOFT)*. IEEE, 2021, pp. 1–5.

[22] B. A. Muse, M. Rahman, C. Nagy, A. Cleve, F. Khomh, and G. Antoniol, "On the prevalence, impact, and evolution of SQL code smells in data-intensive systems," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2020, pp. 327–338.

[23] R. Oberhauser and C. Pogolski, "VR-EA: Virtual reality visualization of enterprise architecture models with ArchiMate and BPMN," in *International Symposium on Business Modeling and Software Design (BMSD)*. Springer, 2019, pp. 170–187.

[24] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microvision: Static analysis-based approach to visualizing microservices in augmented reality," in *IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 49–58.

[25] A. S. Abdelfattah, T. Cerny, D. Taibi, and S. Vegas, "Comparing 2D and augmented reality visualizations for microservice system understandability: A controlled experiment," in *IEEE/ACM International Conference on Program Comprehension (ICPC)*. IEEE, 2023, pp. 135–145.

[26] S. Ardigò, C. Nagy, R. Minelli, and M. Lanza, "Visualizing data in software cities," in *IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2021, pp. 145–149.

[27] ——, "M3tricity: Visualizing evolving software & data cities," in *ACM/IEEE International Conference on Software Engineering (ICSE): Companion Proceedings*. ACM, 2022, pp. 130–133.

[28] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh, "Using service dependency graph to analyze and test microservices," in *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2018, pp. 81–86.

[29] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle, "MicroART: A software architecture recovery tool for maintaining microservice-based systems," in *IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 298–302.

[30] B. Mayer and R. Weinreich, "An approach to extract the architecture of microservice-based software systems," in *IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 2018, pp. 21–30.

[31] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*. Elsevier, 2003, pp. 364–371.