DENIM: Exploring Data Access in Microservices

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

***Abstract*—Adopted by companies such as Netflix, Amazon, and Spotify, the microservices architectural style is now well established. Aimed at facilitating software evolution, it is renowned for modularizing a software system into microservices, implemented in various technologies. Regarding databases, practitioners opt for *polyglot persistence*: Each microservice is responsible for its own database(s). This influences how the architecture is implemented. The decoupling and heterogeneity of microservices and their databases spread data access points throughout the codebase, complicating program comprehension and code-data co-evolution. Developers' feedback reveals their struggles to obtain a holistic view of data access in such architectures.**

We present DENIM, a tool that enables users to identify data access points in microservices and visualize them in an interactive treemap. Using real microservice applications, we illustrate how this tool can be used for software evolution tasks.

https://figshare.com/s/6f1d970b87b7ebce939f?file=54914249

***Index Terms*—microservices, databases, evolution, DENIM**

I. INTRODUCTION

Introduced in 2011, microservices have since increased in popularity, from open source to industrial projects. This architectural style is now adopted by leading companies such as Netflix, Amazon, and Spotify [1]–[3]. Facilitating software evolution is one of the main motivations for adopting microservices, thanks to their modularity, heterogeneity, and runtime interactions via lightweight protocols [1]–[4].

Paradoxically, recent studies [5] suggest that this promise is not being fully realized, particularly regarding database management. Microservices have influenced the way databases are integrated. Rather than sharing a single *one-size-fits-all* database, each microservice manages its own database(s). This follows the polyglot persistence [3] and the *database-per-microservice* patterns [5]. As a side effect, modularity, heterogeneity, and dynamic exchanges spread data access points throughout the codebase. Consequently, practitioners encounter difficulties in recovering holistic views of the architecture [6]–[10], which are essential for understanding and thus for supporting software evolution tasks (*e.g.,* impact analysis).

To fill this gap, we present DENIM (Database Evolution Nudge In Microservices), a tool to identify data access points in microservices, leveraging static analysis and natural language processing [11]. It visualizes a given microservices architecture in an interactive treemap to support developers in understanding and gaining insights on the data access layers and technologies [12], especially for evolution purposes.

II. DENIM

DENIM provides developers with a suite of tools for downloading, reverse engineering, and visualizing microservices.

Downloading. First, it helps to download, in one shot, using *Git*, one or several applications composing a microservices architecture, possibly spread across multiple, distributed, and heterogeneous Git repositories. This tool is available as an API with dedicated endpoints.

Reverse Engineering. Based on the downloaded codebase(s), it automatically mines and identifies data access code fragments (*i.e.,* API and database invocations) using a custom heuristic-based static analyzer [11], [13] built upon *CodeQL* [14] for abstract syntax tree inspection. It leverages natural language processing techniques (*e.g.,* lemmatization) using WinkJS [15] and Natural [16] to characterize the fragments. Each one is enhanced with additional details, such as the database technology used, the type of operation, the specific method employed, and, when available, a sample of the data object or value affected by the operation. Fragments may be linked to one or more data concepts (*i.e.,* data entities extracted from source code analysis). The static analyzer generates a comprehensive and detailed analysis report that serves as the basis for creating the visualization. This tool is available as an API with dedicated endpoints.

Visualizing. Once the analysis report is uploaded to the user interface (see Section II-A), an interactive treemap visualization is generated to present the microservices architecture to the user [12]. Using a bin-packing *First Fit Decreasing* heuristic algorithm,[1] this scalable vectorial visualization gathers the representation of the containment hierarchy of repositories, directories, and files in a 2D compact space. It focuses on data access code fragments using icons representing types of operations and customizable colors. The interactive treemap is overlaid with on-demand tooltips giving detailed information collected for characterizing the fragment. Finally, the treemap is navigable and explorable through features like zooming, colorization, and concept location, as detailed in Section II-B. This tool is available as a web application.

A. User Interface

Figure 1 presents the user interface of DENIM. At the top Ⓐ, we see the uploaded analysis report. In this case, it is named "`Overleaf (Nov 4 2024).json`". The button on the right allows the user to consult the raw analysis report. At the bottom-center Ⓑ, we see the generated interactive treemap related to the analysis report. In between, we find a collapsible section with color pickers Ⓒ, Ⓓ, and selectors Ⓔ for filtering and highlighting features.

[1] **`https://www.npmjs.com/package/bin-pack`**

Fig. 1. User interface of DENIM.

Fig. 2. Technology distribution, heterogeneous (left) vs. homogeneous (right).

Fig. 3. Concept highlighting.

B. Features & Use Cases

1) Exploration: The treemap is movable and zoomable through mouse drag and scroll. This feature helps developers navigate and explore the treemap regardless of the size of the system. The compact 2D presentation and SVG format make the treemap easy to scale.

2) Technology breakdown: The interactive treemap is coupled with technology-dedicated color pickers (Figure 1 Ⓒ). The user can select a hue through the color picker sliders. Different colors help developers understand at a glance the distribution of technologies within the architecture, at macro (*i.e.,* folder, microservice, repository) and micro levels (*i.e.,* file and code fragment), as illustrated in Figure 2. Developers can easily draw conclusions, for instance, about technology prevalence, and perform tasks like determining whether components are organized homogeneously or heterogeneously, identifying polyglot components, localizing data-intensive areas, comparing elements within the architecture, etc.

3) Operation distribution: Similarly, operation types have color pickers Ⓓ that update the interactive treemap. In the same way, developers can highlight the distribution of operations, which gives insights into data-intensive parts or less sensitive areas like read-only microservices.

4) Concept location: The treemap helps highlight code fragments associated with a particular concept. Selecting a concept in the field Ⓔ the treemap is automatically updated by making opaque all code fragments that are not associated with the selected concept, as illustrated in Figure 3. This feature helps to assess how a concept is distributed throughout the system, which is valuable for evaluating the potential impact of modifying the data structure of that concept (impact analysis).

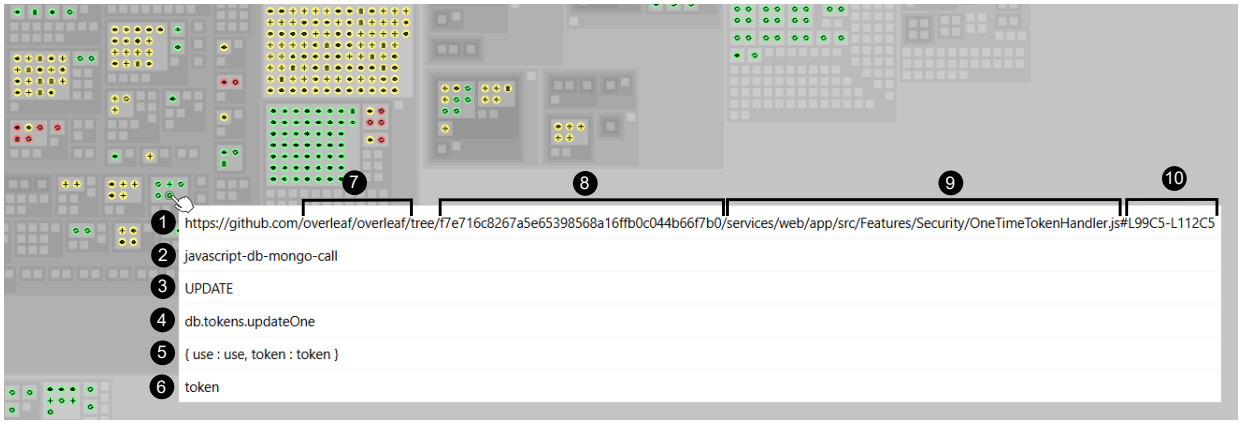

Fig. 4. Code fragment details.

5) On-demand details: Additional details are available when hovering over a box or code fragment to support interactive navigation and to provide additional information. As depicted in Figure 4, a tooltip appears and provides information on the code fragment's name and location (including the full path) ①, technology type ②, operation type ③, extracted method name ④, extracted argument definitions ⑤, identified concept ⑥, repository name ⑦, version of the source code (via the *git* hash) ⑧, the repository, folders, and file hierarchy ⑨, and finally the precise location of the code fragment inside the file, through line and column numbers ⑩.

6) In-context view: Thanks to the availability of the complete location, a simple click on a code fragment is enough to lead the user directly to the GitHub repository to see the code fragment in its context, which helps with further analysis.

Concluding remarks. The presented features help developers to understand systems, which is essential for preparing evolution tasks. Among relevant tasks we can note: Impact analysis by concept location, quality assessment by examining code structure and the distribution of technologies and operations, version comparison through the shapes and color patterns generated by the treemap, and anti-pattern identification (*e.g.,* lack of separation of concerns) by treemap inspection.

C. Architecture

DENIM is developed as a microservices architecture composed of four units (see Figure 5). First, *Downloading* fulfills the task to retrieve one or several microservice repositories. The "`/git`" route allows users to send a list of git repositories with their hashes and receive in return a *zip* file containing the entire architecture. This zip file is then sent to *Reverse Engineering* through the route "`/static`", which generates a database of the abstract syntax tree, using CodeQL [14]. Thanks to custom heuristics built on top of CodeQL, *Reverse Engineering* identifies data access code fragments and returns the analysis report in JavaScript Object Notation (JSON) format. Via the route "`/treemap`", the analysis report is sent to *Visualization*, which uses it to generate the treemap layout with the bin-packing *First Fit Decreasing* heuristic algorithm. Finally, *Web* handles user interaction and treemap rendering.

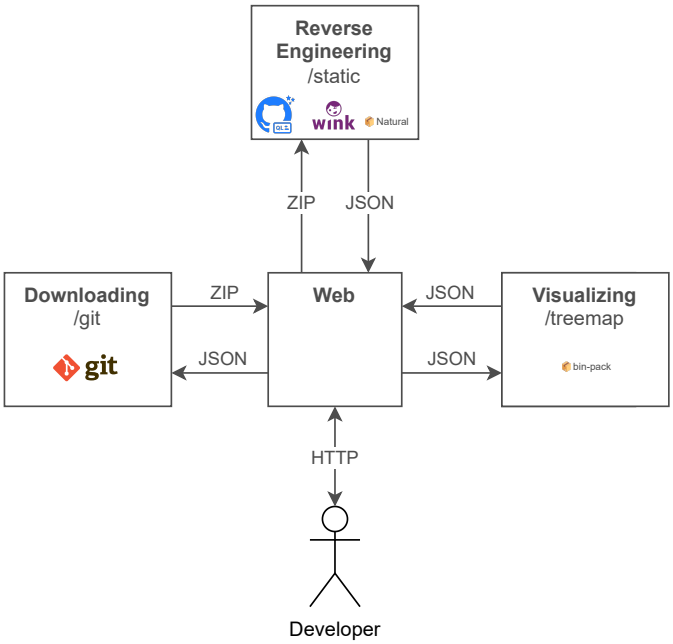

Fig. 5. Architecture of DENIM.

D. Model

The backbone of DENIM is its underlying model presented in Figure 6. A microservices architecture is represented as a set of repositories (`Repository`) subdivided into directories (`Directory`) and files (`File`). Finally, the terminal nodes of the chain are the collection of code fragments (`CodeFragment`). Each code fragment is enriched with additional details extracted during static analysis, such as lines of code (LoC), the technology used for data access, the associated Create, Read, Update, or Delete (CRUD) operation, the specific Object-Relational Mapping (ORM) method employed, and, when available, a sample of the data objects or values affected by the operation. Finally, each code fragment may be linked to one or more extracted data concepts (`Concept`).

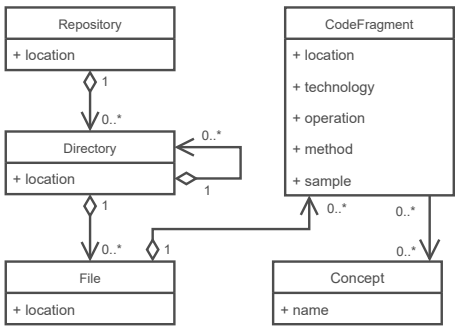

Fig. 6. Model of DENIM.

III. EXAMPLES

In order to assess that DENIM is able to support a range of scales and complexities, we selected 10 real-world microservices applications that we downloaded, analyzed, and visualized through our tool, as depicted in Figure 7. We selected those applications from GitHub by applying the following criteria: Each of them was updated at least once since January 1, 2015, is at least 5 MB in size, has more than 100 stars, and contains at least 100 commits. To ensure the relevance of each architecture according to microservices standards, we manually inspected key files (*e.g.,* Docker Compose, README) and verified their suitability. The generated treemaps for these architectures are included in our replication package (Section VI).

IV. RELATED WORK

Prior work covers tools for recovering and visualizing microservices architectures *or* analyzing database access, but none address both together.

Granchelli et al. introduce *MicroART*, a tool combining static and dynamic analysis to generate, based on configuration files and traces, a model representation of a microservices architecture [17]. *Rahman et al.* also exploit configuration files such as Docker in their tool *MicroDepGraph* for analyzing service dependencies and visualizing them as a dependency graph [18]. Still using graph representations, *Baltes et al.* present a visualization tool for understanding large-scale industrial service-oriented architectures (*e.g.,* REST APIs) to facilitate maintenance tasks [19]. *Silva et al.* opt for graphs too in their tool μViz, using runtime traces to provide logical, physical, trace, and workflow views of microservices in a dashboard [20]. *Mayer et al.* propose and evaluate with 15 developers a dashboard visualizing static and dynamic dependency graphs [21]. *MicroKarta*, another dashboard tool by *Manglaras et al.* [22], is the result of a study exploring the challenges of understanding microservice structures in an industrial context [23]. *Fittkau et al.* introduce another dimension with *ExplorViz*, a hierarchical 3D visualization tool for large software architectures, including microservices [24]. Still in 3D, combining augmented reality, *Cerny et al.* present *Microvision*, a tool for visualizing microservices architectures and supporting navigation across abstraction levels [25].

Similarly, *Oberhauser and Pogolski* propose *VR-EA*, using virtual reality to document and visualize large, dynamic enterprise solutions, adapted to microservices [26]. Several studies [6], [8]–[10], [24] mention industrial tools: Amazon X-Ray,[2] Simianviz,[3] Zipkin,[4] Kiali,[5] Instana,[6] Jaeger UI.[7]

On databases, *Nagy and Cleve* introduce *SQLInspect* to statically extract and assess SQL queries from Java [27]. Similarly, *Liu and Chen* propose *SLocator*, combining static analysis and information retrieval to locate SQL queries generated by JPA ORM [28]. Adding a visual 3D city metaphor to visualize the evolution of software and databases, *Ardigò et al.* present *M3tricity* [29] and *M3tricity2* [30]. However, both versions of the tool are not specific for microservices architectures.

V. CONCLUSIONS

Microservice architectures have established themselves in the software development landscape for over a decade, from open source projects to major industrial initiatives led by prominent companies such as Netflix, Amazon, and Spotify.

While these architectures are intended to facilitate software evolution through modularity, heterogeneity, and dynamic interoperability, significant challenges remain. Notably, the distributed and heterogeneous nature of microservices and their databases affects and complicates the recovery of the holistic view that developers need to understand a microservices architecture before evolving it.

In this paper, we presented DENIM, a tool designed to address such challenges by enabling the identification and visualization of data access points across microservices. By leveraging static analysis and interactive treemaps, we provide developers with insights to better understand microservice architectures. DENIM facilitates software evolution tasks such as impact analysis, technology assessment, and concept localization, at scales ranging from small to large and complex, with the potential to generalize support of code and data co-evolution beyond the microservices architectural style.

VI. REPLICATION PACKAGE AND REPOSITORY

To ensure verifiability and replicability of our work, the artifacts used to demonstrate our tool (🗃) and the tool itself (</>) are publicly available as open source.

🗃 https://figshare.com/s/6f1d970b87b7ebce939f
</> https://github.com/DatabaseEvolutionNudgeInMicroservices/denim

ACKNOWLEDGMENTS

Research supported by the SofinaBoël Fund for Education and Talent and the Federation Wallonie-Bruxelles (ARC project RAINDROP), and by the Swiss National Science Foundation through the project "FORCE" (SNF project 232141).

[2] **https://aws.amazon.com/xray/**
[3] **https://simianviz.surge.sh/**
[4] **https://zipkin.io/**
[5] **https://kiali.io/**
[6] **https://www.ibm.com/en-en/products/instana**
[7] **https://github.com/jaegertracing/jaeger-ui**

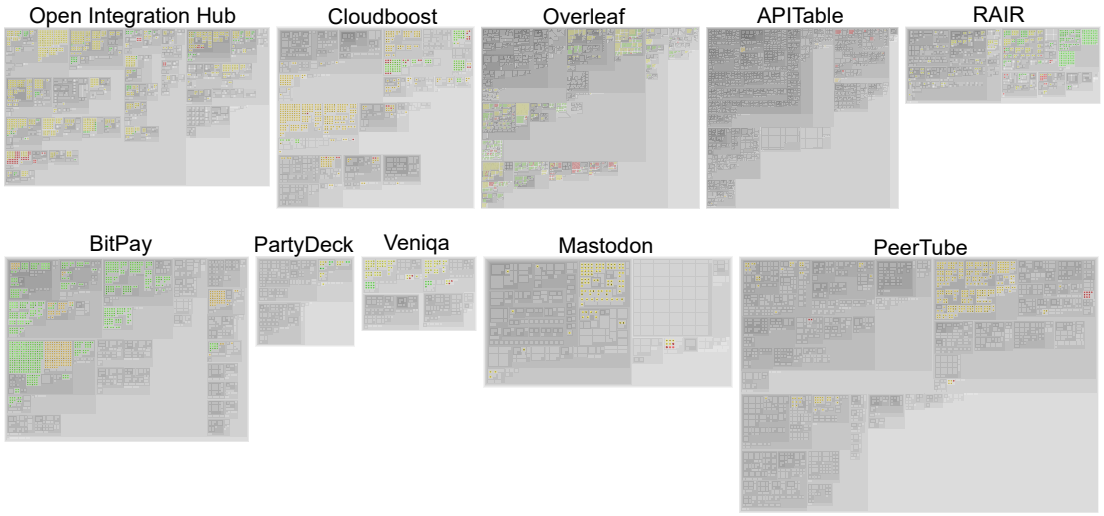

Fig. 7. Treemaps generated for real-world microservices applications.

REFERENCES

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

[2] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2021.

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

[4] M. André, "Automated Database Schema Evolution in Microservices," in *Conference on Very Large Data Bases (VLDB): PhD Workshop Track*, vol. 3452. CEUR-WS, 2023, pp. 37–40.

[5] 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.

[6] 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.

[7] 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

[8] 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.

[9] 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.

[10] 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.

[11] M. André, E. Rivière, and A. Cleve, "Data Access-centered Understanding of Microservices Architectures," in *IEEE International Conference on Software Architecture (ICSA): NEMI Track*. IEEE, 2025.

[12] M. André, M. Raglianti, A. Cleve, and M. Lanza, "Understanding Data Access in Microservices Applications Using Interactive Treemaps," in *IEEE/ACM International Conference on Program Comprehension (ICPC): ERA Track*. IEEE/ACM, 2025.

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

[14] GitHub, "About CodeQL — CodeQL," https://codeql.github.com/docs/codeql-overview/about-codeql/, 2024.

[15] graype systems, "winkJS — NLP, Machine Learning & Stats in Node.js," https://winkjs.org/, 2024.

[16] NaturalNode, "Natural," https://naturalnode.github.io/natural/, 2024.

[17] 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.

[18] M. I. Rahman, S. Panichella, and D. Taibi, "A Curated Dataset of Microservices-based Systems," in *Joint of the Summer School on Software Maintenance and Evolution (SSSME)*. CEUR-WS, 2019, pp. 1–9.

[19] S. Baltes, B. Pfitzmann, T. Kowark, C. Treude, and F. Beck, "Visually Analyzing Company-wide Software Service Dependencies: An Industrial Case Study," in *Working Conference on Software Visualization (VISSOFT)*. IEEE, 2023, pp. 23–27.

[20] S. Silva, J. Correia, A. Bento, F. Araujo, and R. Barbosa, "μViz: Visualization of Microservices," in *International Conference Information Visualisation (IV)*. IEEE, 2021, pp. 120–128.

[21] 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.

[22] O. Manglaras, A. Farkas, P. Fule, C. Treude, and M. Wagner, "MicroKarta: Visualising Microservice Architectures," in *International Conference on the Foundations of Software Engineering (FSE)*, 2024, pp. 607–611.

[23] ——, "Problems in Microservice Development: Supporting Visualisation," in *Working Conference on Software Visualization (VISSOFT)*. IEEE, 2023, pp. 62–72.

[24] F. Fittkau, A. Krause, and W. Hasselbring, "Software Landscape and Application Visualization for System Comprehension with ExplorViz," *Information and Software Technology*, vol. 87, pp. 259–277, 2017.

[25] 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.

[26] 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.

[27] C. Nagy and A. Cleve, "SQLInspect: A Static Analyzer to Inspect Database Usage in Java Applications," in *International Conference on Software Engineering (ICSE)*, 2018, pp. 93–96.

[28] W. Liu and T.-H. Chen, "SLocator: Localizing the Origin of SQL Queries in Database-Backed Web Applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 6, pp. 3376–3390, 2023.

[29] S. Ardigò, C. Nagy, R. Minelli, and M. Lanza, "M3triCity: Visualizing Evolving Software & Data Cities," in *ACM/IEEE International Conference on Software Engineering (ICSE): Companion Proceedings*. ACM, 2022, pp. 130–133.

[30] ——, "Visualizing Data in Software Cities," in *IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2021, pp. 145–149.