Immersive Software Archaeology: Exploring Software Architecture and Design in Virtual Reality

Adrian Hoff*, Christoph Seidl*, Michele Lanza†
*IT University of Copenhagen, Denmark — †Software Institute @ USI Università della Svizzera italiana, Switzerland

***Abstract*—Comprehending large-scale software systems is a challenging and daunting task, particularly when only source code is available. While software visualization attempts to aid that process, existing tools primarily visualize a system's structure in terms of files, folders, packages, or namespaces, neglecting its logical decomposition into cohesive architectural components.**

We present the tool Immersive Software Archaeology (ISA) which (i) estimates a view of a system's architecture by utilizing concepts from software architecture recovery and (ii) visualizes the results in virtual reality (VR) so that users can explore a subject system interactively, making the process more engaging. In VR, a semantic zoom lets users gradually transition between architectural components of different granularity and class-level elements while relationship graphs let users navigate along connections across classes and architectural components.

We present results from a controlled experiment with 54 participants to investigate the usefulness of ISA for assisting engineers with exploring an unfamiliar large-scale system compared to another state-of-the-art VR approach and an IDE.

***Video Demonstration*—https://youtu.be/Fl_SsT13l4k**

***Index Terms*—Software Visualization, Software Architecture Visualization, Software Comprehension, Software Re-Engineering**

I. Introduction

Software archaeology is the process of recovering knowledge on an unfamiliar (legacy) software system using the artifacts available, which commonly constitute only source code [1], [2]. A relevant activity is recovering a subject system's architecture which - when conducted based on source code alone - is tedious, challenging, and error-prone [2], [3]. By explicitly encoding information on a system's structure, behavior, and/or evolution via a visual metaphor, software visualization supports various tasks such as estimating or relating architectural components.

The extent of existing tool support is limited: current 3D/VR software visualizations (i) rigidly locate users on an abstraction level only slightly above source code while (ii) visualizing a system's architecture in terms of its organization in folders, namespaces, or other "physical" structures.

We present the tool Immersive Software Archaeology[1] (ISA), which supports software archaeologists with recovering knowledge on an unfamiliar large-scale system by visualizing its architecture and design in immersive virtual reality (VR) based on its source code alone. Throughout the paper, we use the term "design" to refer to elements on class level and below, which are represented explicitly in source code, and the term "architecture" to refer to logical elements more abstract than class level, which are not represented explicitly in source code.

[1]https://gitlab.com/immersive-software-archaeology

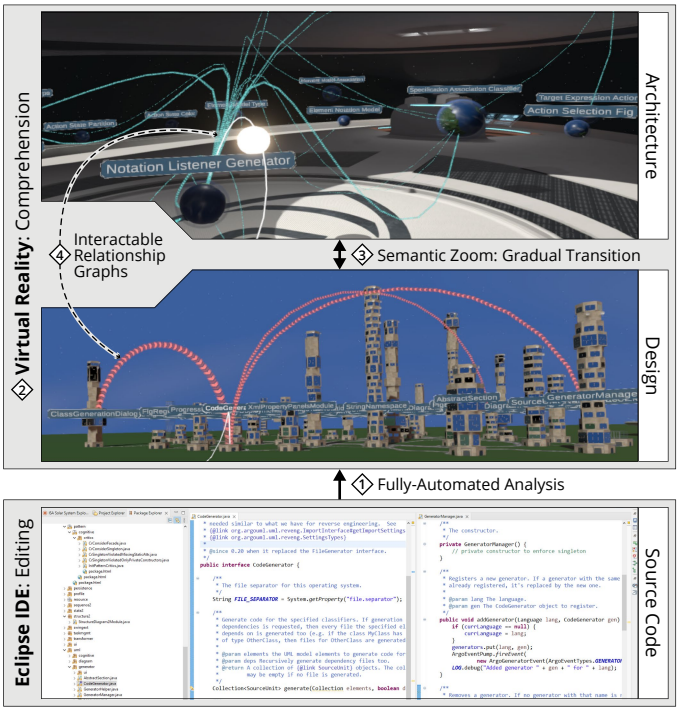

Fig. 1. Overview of ISA from a user's perspective: After analyzing a subject system in the IDE, users explore its architecture and design in VR. They can transition between abstraction levels via a semantic zoom and navigate along relationships via an interactive graph.

We present four core features of ISA (cf. Figure 1):

1. ISA employs a fully-automated relationship-based clustering method to analyze a system's architecture in terms of cohesive architectural components.
2. ISA presents the results of its analysis via an immersive, interactive metaphor in virtual reality.
3. ISA implements a semantic zoom that lets users navigate seamlessly along abstraction levels from architecture down to design level (and vice versa).
4. ISA visualizes relationships between elements (i.e., classes and architectural components) via interactive relationship graphs, among and across abstraction levels.

We evaluated ISA in a controlled experiment with 54 participants [4], comparing it to an existing state-of-the-art VR software visualization and the Eclipse IDE. Our results show that ISA provides better access to information while fostering users' ability to relate software elements (classes, architectural components, etc.) with one another.

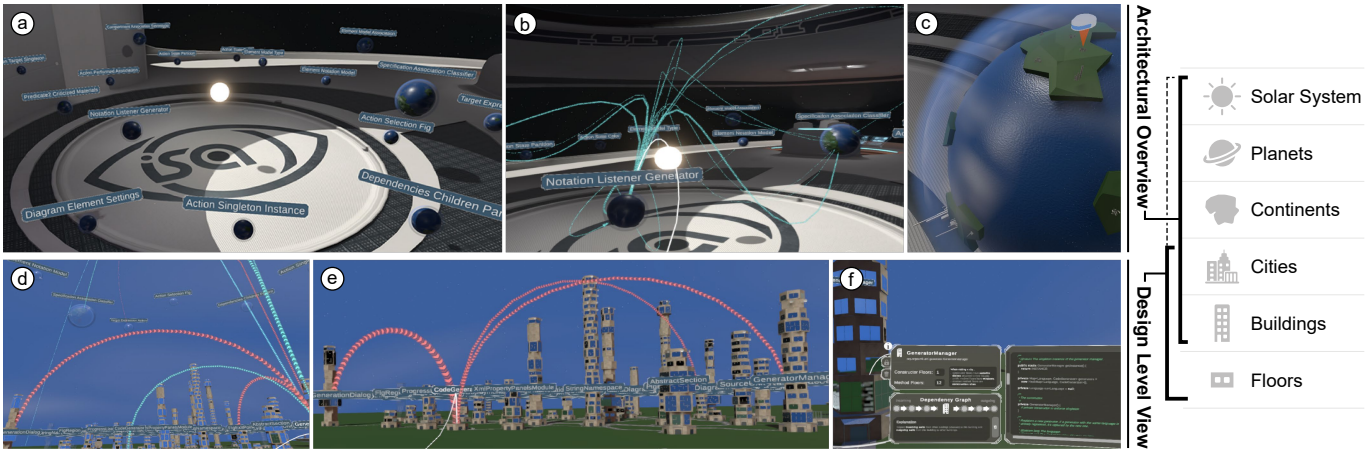

Fig. 2. Screenshots of ISA's VR application: a system's hierarchy of architectural components is visualized as a solar system with planets, hierarchies of continents, and cities, while design-level elements (classes, methods, etc.) are visualized as buildings and floors. A semantic zoom enables users to gradually transition between abstraction levels, while relationship graphs provide navigation along relationships.

II. State of the Art

Many software visualization tools exist that present a system's architecture [5], [6], both in 2D (e.g., Polymetric Views [7], Samoa [8], Softwarenaut [9], Beck et al. [10]) and 3D (e.g., Balzer and Deussen [11], IslandViz [12], ExplorViz [13], CityVR [14], CodeCity [15]). Experiments showed that using a well-constructed 3D software visualization is more effective than using an IDE in terms of correctness and completion time for software comprehension tasks [15]. In addition, 3D visualizations have advantages over 2D visualizations in terms of spatial memory [16]. 3D metaphors become further effective (task completion time and accuracy) when using the immersive capabilities of VR to allow for a more natural interaction with a 3D scene as compared to a standard-screen representation [17], [18].

Metaphors for 3D software visualization can be divided into abstract metaphors (e.g., 3D graphs [11], [19], [20]) and real-world metaphors (e.g., city metaphor [14], [15], [17], island metaphor [12], or solar metaphors [21]–[24]). A metaphor has to be expressive enough to describe different abstraction levels without breaking while letting users transition between these.

We observe two major shortcomings in existing 3D (and thus VR) software visualization tools with regard to presenting a system's architecture. Many tools [12]–[15] rigidly lock their users on an abstraction level slightly above the level of (textual) source code, where they focus on code metrics (such as lines of code) instead of providing an overview of a system's architecture. Furthermore, even existing tools focusing on a system's architecture commonly visualize merely folders, packages, or namespaces [5], although these "physical" structures can deviate heavily from a logical arrangement into architectural components (potentially cutting across folders, packages, and namespaces) – especially after evolution cycles in a legacy software system. This leaves open potential for supporting users in relevant software archaeology tasks such as identifying and relating architectural components.

III. Immersive Software Archaeology

Our tool Immersive Software Archaeology (ISA) is comprised of two parts, i.e., an extensible platform of Eclipse[2] plugins for the automated analysis of a subject system's architecture and an immersive VR application developed with the Unity 3D engine[3] for the exploration of a system based on a previously conducted analysis. Once having analyzed a system in Eclipse, users can enter ISA's VR visualization and inspect the analyzed system via a real-world metaphor (see Section III-B) with planets, continents, cities, and buildings, where buildings represent design-level elements (i.e., on the level of classes) and cities, continents, and planets represent higher-level architectural components (i.e., based on a logical structure that is not directly visible in code). From within the VR visualization, users can access a backchannel to the IDE. That is, they can open windows in the IDE from within VR that display the source code of visualized files, e.g., to then directly perform changes in the IDE.

A. Automated Architecture and Design Analysis in Eclipse

ISA provides a fully-automated analysis of a software system that takes as input the source code of a subject system and generates as output a coherent model of its architecture and design. Along with our tool, we provide an implementation for the analysis of Java software systems. ISA's analysis consists of two subsequent steps.

First, to provide users with design-level information, ISA's analysis extracts design-level information explicit in source code, i.e., the structure of classes, interfaces, enums, etc. with their members and references between them. In addition, ISA calculates metrics for design-level elements such as the cognitive complexity of methods [25]. An output model is populated with the analysis results to capture a view on an entire subject system's design-level structure.

[2]https://www.eclipse.org/
[3]https://unity.com

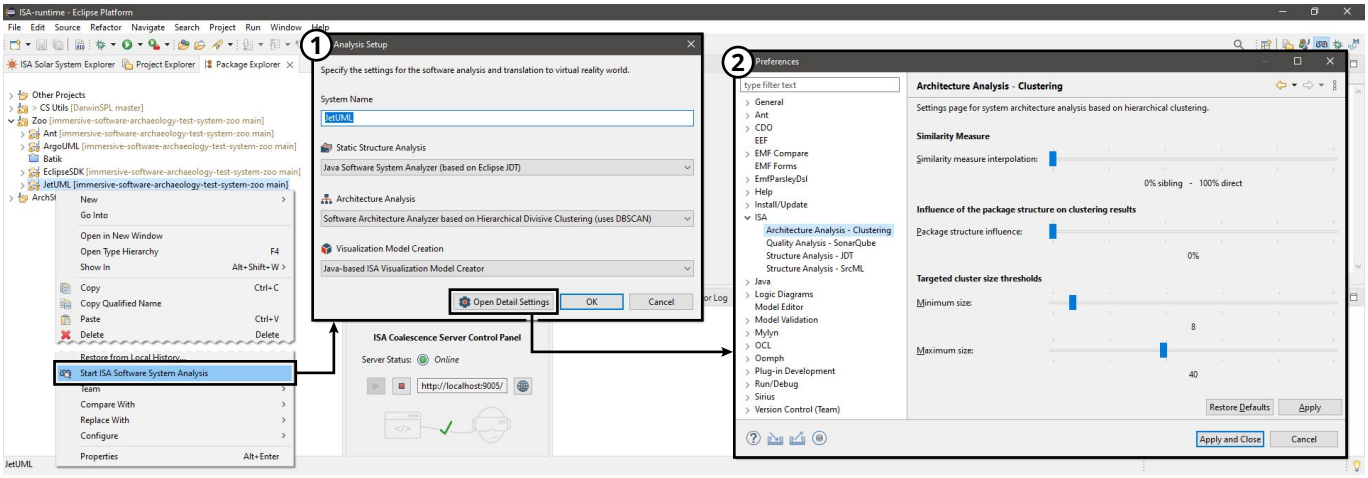

Fig. 3. Screenshots of ISA's extensible Eclipse plugins, where (1) users choose between different analysis plugins and (2) optionally change detail settings for these. Users can implement their own ISA analysis plugins and make them available for selection by registering with a central analysis core plugin.

Second, to provide users with an overview of a system's architecture, we provide a software architecture recovery that estimates a decomposition of the previously analyzed design-level elements into a hierarchy of architectural components (i.e., coherent units of functionality). To achieve this, ISA employs a software clustering technique based on the references between classes as described in [4]. The resulting output model encompasses all design-level elements of a system, organized into a hierarchy of cohesive architectural components. When starting ISA's VR application, this model is encoded into the high-level structure of the visualization, determining how planets, their hierarchies of continents, city layouts, as well as the buildings with their individual floors are generated.

On a technical level, ISA's overall analysis process described above is implemented as an extensible platform of Eclipse plugins, each registering with a central analysis core plugin. Developers can implement custom analysis functionality in their own plugins, e.g., to integrate further target languages (for instance, based on SrcML[4] [26] or Tree-Sitter[5]), or to employ an alternative clustering strategy. An analysis configuration dialog (① in Figure 3) lets users choose between the available plugins for each step of the analysis. Furthermore, ISA provides detail settings for its plugin, e.g, to adjust parameters for the software architecture recovery procedure (② in Figure 3).

We define the above described models for capturing a system's architecture and design via the Eclipse Modeling Framework[6] (EMF). By generating source code infrastructure from these models in both Java (for use with Eclipse) and C# (for use with Unity), they serve two purposes. For one, they act as interfaces between the different analysis steps in the IDE, e.g., between design analysis and architecture analysis as described above. For another, they serve as input during the VR visualization's initialization.

[4]https://www.srcml.org
[5]https://tree-sitter.github.io/tree-sitter
[6]https://www.eclipse.org/modeling/emf

B. Virtual Reality Exploration of Architecture and Design

ISA's VR visualization employs an immersive real-world metaphor representation of a subject system, where users travel through and interact with its architecture and design. By explicitly representing architectural components as first-level structural elements, ISA facilitates users' ability to gain an architectural overview of a system. A semantic zoom lets users gradually transition from this architectural overview to architectural components of lower abstraction levels, down to design-level elements such as classes and their members. Relationship graphs visualize calls and references between classes and architectural components in the context of the user's current abstraction level.

We implement the ISA VR application with the Unity 3D engine[3], building upon the SteamVR[7] library to support major VR headsets currently available. We illustrate its visualization technique with structures known from object-oriented programming (e.g., classes and interfaces). However, ISA is able to visualize systems implemented in any programming languages with an organization of methods/functions in modules (such as classes) and a hierarchical organization of these.

1) Architecture-Level Overview: To foster users' ability to explore and comprehend the architecture of an unfamiliar system, ISA visualizes architectural components as first-level structures in its visualization: It employs a solar system metaphor where the hierarchically organized architectural components recovered previously (see Section III-A) are represented as planets (ⓐ, ⓑ, ⓒ in Figure 2). Corresponding to the hierarchical organization of the architectural component it represents, each planet contains hierarchically organized pieces of land [4] – similar to how continents, countries, and regions are hierarchically organized in the real world. On the lowest abstraction level, architectural components are represented as cities consisting of buildings which each represent a class-level element (classes, interfaces, enums, etc.) of the subject system.

[7]https://www.steamvr.com/en

Buildings consist of floors, each representing a method or constructor with the number of expressions determining the floor's height and the cognitive complexity [25] determining the floor's diameter. By using these metrics to determine the visual appearance of each building's floors (and thus the building itself), ISA provides users with an impression of the represented class' structure without having to read its source code.

2) Semantic Zoom: To provide users with access to information on a system while not overwhelming them with details, ISA implements a semantic zoom that displays visual elements (such as buildings) context-sensitively, i.e., with more or less detail depending on the abstraction level of users. When starting the ISA VR visualization, users are initially located in an interactive room-scale overview of a system's architecture (ⓐ, ⓑ, ⓒ in Figure 2), where they can inspect and interact with its architectural components in form of planets, hierarchies of continents, and cities. On this abstraction level, cities on planets consist of simplified versions of their buildings that visually average the metrics for their floors, cf. ⓒ in Figure 2. Users can interact with planets by grabbing and repositioning them (e.g., to locate two planets close to one another to indicate coherence) and opening information canvases to inspect further details (e.g., the number of buildings on a planet).

Users can change their perspective from the architectural overview to a detailed city view (ⓓ, ⓔ, ⓕ in Figure 2), where buildings present each method and constructor of the represented classes explicitly with metric values visualizing structural properties. Within a city, users can freely inspect design-level structures, open information (ⓕ in Figure 2) on classes, and read their source code. This semantic zoom enables users to gradually change their perspective from an overview of a system architecture down to design level and vice versa, following Shneiderman's mantra "Overview first, zoom and filter, then details on-demand" [27].

3) Interactable Relationship Graphs: To support users in gaining an overview of the relationships among and across architectural components and classes, ISA encompasses interactable relationship graphs. These visualize relationships between software elements (i.e., classes or architectural components) as curved lines between their visualization counterparts, with animated textures indicating the flow direction of information. To visualize relationships on architecture level (e.g., between two architectural components represented as planets), ISA automatically aggregates and bundles references of architectural components, i.e., all method calls, field accesses, and type references (incl. inheritance). Users can interact with these relationship graphs via a VR user interface (ⓕ in Figure 2) that enables them to highlight forward references (connecting to all referenced elements) and backward references (connecting all referencing elements), including their transitive closures. Thereby, ISA aids users in tasks such as identifying module boundaries, e.g., by investigating whether a selected class is strongly connected with a selection of other classes.

IV. Evaluation

We evaluated ISA in a controlled experiment with 54 participants [4], where we compared its ability to support engineers with performing software archaeology tasks with (i) an existing state-of-the-art VR software visualization (CityVR [14]) and (ii) an IDE (Eclipse) as a baseline. We split the participants into three groups, each using a different tool (i.e., ISA, CityVR, or Eclipse), and let them explore a real-world legacy subject system via five software archaeology tasks (identical across participants and tools) on accessing and relating information, e.g., finding a part of the system given a description of its functionality. Thereby, we simulated a step-wise exploration process in which participants established an understanding of the subject system, while measuring and observing their behavior and approaches toward solving each task. Detailed task descriptions, results, and a replication package are available in the study's online appendix[8].

Our results [4] show that, in comparison with the other tools, ISA's organization of a system's classes in cohesive architectural components provides better access to information because interrelated classes (and components) are likely to be part of the same structure (e.g., buildings in the same city) and can thus be explored side-by-side. In combination with ISA's interactable relationship graphs, participants were able to estimate components more sensibly in terms of size and content as compared to the state-of-the-art VR visualization and the IDE. Conversely, ISA's interactable relationship graphs caused a tendency for participants to not investigate relations between elements as thoroughly as in the IDE – that is, beyond seeing a line between the respective visual elements (e.g., buildings in a city). However, ISA's combination of the interactable relationship graphs with its semantic zoom, especially its ability to agglomerate relationships to architecture level, showed to be useful particularly for inexperienced engineers who, when using one of the other tools, faced difficulties with reconstructing relationships even on class level.

V. Conclusion and Future Work

We presented the tool Immersive Software Archaeology (ISA) which supports users in recovering knowledge of an unfamiliar software system via an immersive VR visualization of the system based on only its source code. To achieve this, ISA provides (i) a fully-automated configurable and extensible architecture recovery method and, based on that, (ii) an interactive VR visualization that lets users immersively explore and interact with a subject system's architecture and design. A semantic zoom in VR lets users transition between abstraction levels of architecture and design, while interactable relationship graphs provide an overview of interactions between elements, e.g., between architectural components visualized as cities on different planets, between classes visualized as buildings on the same planet, etc.

[8]https://gitlab.com/immersive-software-archaeology/publication-vissoft22

ACKNOWLEDGEMENTS

Hoff and Seidl are supported by the DFF (Independent Research Fund Denmark) within the project "Immersive Software Archaeology (ISA)" (0136-00070B). Lanza is supported by the Swiss National Science Foundation (SNSF) through the project "INSTINCT" (Project No. 190113).

REFERENCES

[1] G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz, "An empirical approach to software archaeology," in *ICSM 2005*, 2005.

[2] H. Sneed and C. Verhoef, "Re-implementing a legacy system," *Journal of Systems and Software*, vol. 155, 2019.

[3] A. K. Gahalaut, "REVERSE ENGINEERING: AN ESSENCE FOR SOFTWARE RE-ENGINEERING AND PROGRAM ANALYSIS." *International Journal of Engineering Science and Technology*, vol. 2, p. 9, 2010.

[4] A. Hoff, L. Gerling, and C. Seidl, "Utilizing software architecture recovery to explore large-scale software systems in virtual reality," in *VISSOFT 2022*, 2022.

[5] M. Shahin, P. Liang, and M. A. Babar, "A systematic review of software architecture visualization techniques," *Journal of Systems and Software*, vol. 94, 2014.

[6] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, and W. Chen, "A systematic literature review of modern software visualization," *Journal of Visualization*, vol. 23, no. 4, 2020.

[7] M. Lanza and S. Ducasse, "Polymetric views - a lightweight visual approach to reverse engineering," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, Sep. 2003.

[8] R. Minelli and M. Lanza, "SAMOA – A Visual Software Analytics Platform for Mobile Applications," in *2013 IEEE International Conference on Software Maintenance*, Sep. 2013.

[9] M. Lungu, M. Lanza, and O. Nierstrasz, "Evolutionary and collaborative software architecture recovery with Softwarenaut," *Science of Computer Programming*, vol. 79, 2014.

[10] M. Beck, J. Trümper, and J. Döllner, "A visual analysis and design tool for planning software reengineerings," in *VISSOFT 2011*, 2011.

[11] M. Balzer and O. Deussen, "Level-of-detail visualization of clustered graph layouts," in *6th Int. Asia-Pacific Symposium on Visualization*, 2007.

[12] E. F. Heidmann, L. von Kurnatowski, A. Meinecke, and A. Schreiber, "Visualization of Evolution of Component-Based Software Architectures in Virtual Reality," in *VISSOFT 2020*.

[13] W. Hasselbring, A. Krause, and C. Zirkelbach, "Explorviz: Research on software visualization, comprehension and collaboration," *Software Impacts*, vol. 6, 2020.

[14] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "Cityvr: Gameful software visualization," in *ICSME 2017*.

[15] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *ICSE 2011*, 2011.

[16] M. Tavanti and M. Lind, "2d vs 3d, implications on spatial memory," in *INFOVIS 2001*, 2001.

[17] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. Gonzalez-Barahona, and M. Lanza, "Codecity: A comparison of on-screen and virtual reality," *Information and Software Technology*, vol. 153, 2023.

[18] J. Vincur, P. Návrat, and I. Polasek, "Vr city: Software analysis in virtual reality environment," in *QRS-C 2017*, 2017.

[19] O. Greevy, M. Lanza, and C. Wysseier, "Visualizing live software systems in 3d," in *Proceedings of the 2006 ACM symposium on Software visualization*, 2006.

[20] A. Hoff, C. Seidl, M. Lungu, and M. Lanza, "Preparing Software Re-Engineering via Freehand Sketches in Virtual Reality," in *Proceedings of the 39th IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2023. [Online]. Available: https://www.inf.usi.ch/lanza/Downloads/Hoff2023b.pdf

[21] H. Graham, H. Y. Yang, and R. Berrigan, "A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics," 2004.

[22] A. Hoff, M. Nieke, and C. Seidl, "Towards immersive software archaeology: regaining legacy systems' design knowledge via interactive exploration in virtual reality," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.

[23] A. Hoff, L. Gerling, and C. Seidl, "Utilizing Software Architecture Recovery to Explore Large-Scale Software Systems in Virtual Reality," in *2022 Working Conference on Software Visualization (VISSOFT)*. Limassol, Cyprus: IEEE, Oct. 2022, pp. 119–130. [Online]. Available: https://ieeexplore.ieee.org/document/9978380/

[24] R. Oberhauser and C. Lecon, "Virtual Reality Flythrough of Program Code Structures," in *Proceedings of the Virtual Reality Int. Conference - Laval Virtual 2017 on - VRIC '17*, 2017.

[25] G. A. Campbell, "Cognitive complexity: An overview and evaluation," in *Proceedings of the 2018 international conference on technical debt*, 2018.

[26] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcml: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration," in *2013 IEEE International conference on software maintenance*. IEEE, 2013, pp. 516–519.

[27] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The craft of information visualization*, 2003.