Manipulating VR-Native User Interfaces for Software Visualization Customization

Mattia Giannaccari, Marco Raglianti, Michele Lanza
REVEAL @ Software Institute – USI, Lugano, Switzerland

***Abstract*—Software visualization concerns itself with the visual depiction of software systems to facilitate their comprehension. Any visualization approach, whether 2D or 3D or immersive, comes with a plethora of configuration possibilities (*e.g.,* which types of artifacts to visualize and how, which layouts to use). This reflects the complexity of the domain at hand, where manipulating millions of entities pertaining to dozens of different types of artifacts is common. Most visualization tools encode their customizations in the form of view configurations/specifications (in short viewspecs), which are either created declaratively (using DSLs), or through custom user interfaces. In the case of immersive visualization, approaches using such customization facilities are cumbersome, may generate unnecessary context and paradigm switches, and fail to leverage the full potential of modern VR headsets' controllers.**

We present an approach to interactively manipulate the view specifications by depicting them as 3D objects in the immersive space, supporting definition and configuration with an automatic reflection-based mapping of the software domain model under exploration. IVAR-NI**, the tool we developed, incorporates new immersive interaction paradigms (*e.g.,* slot-based selection) and in-object real-time feedback (*e.g.,* preview of the view specification effects) to enhance the usability of this new generation of VR-native interfaces for software visualization customization.**

`https://youtu.be/HsWGtrINtHc`

***Index Terms*—Virtual Reality, VR-Native Interfaces, Software Visualization, VR Interactive Interfaces, IVaR-NI**

I. Introduction

Virtual Reality (VR) headsets reached a maturity that makes them robust enough to be viable for everyday visualization tasks and scientific experiments. From so called serious games [1], to education [2], and medical applications [3], they opened new possibilities to visualize complex data structures [4], [5], and software systems are among the most complex artifacts created by humankind.

Modern software systems can consist of up to hundreds of millions of lines of code [6]. Cisco's IOS-XE software, for example, counts more than 2,200 software components and tens of thousands of source files. It supports hundreds of hardware products for different purposes (*e.g.,* routing, switching, wireless, network management) [6]. Software visualization provides the means to address such complexity and to explore and understand large-scale systems [7].

Immersive VR environments and traditional screen-based dashboards compete for software and data visualization [8]. Custom VR visualizations [9], [10] have been proposed for software development metrics [8], software architecture analysis and recovery [11]–[13], and dependency ecosystems analysis [14]. Nevertheless, these tools are often either predetermined in the way they show the components of the system [11], or provide a simple representation based on well-known metaphors (*e.g.,* the city metaphor [15]) with limited configurability unless leaving the immersive environment.

To remedy to such limitations, we propose a novel approach to manipulate VR-native user interfaces for software visualization customization. IVAR-NI, the tool we developed as a proof-of-concept of our approach, allows to reduce the disconnect from VR by reducing the need to switch back to non-immersive configuration tools, like, for example, 2D Graphical User Interfaces (GUIs), or Domain Specific Languages (DSLs) [16]. Meta-entities in the system (*e.g.,* class prototypes, metrics mappings) are visualized as 3D shapes and manipulated as normal entities to configure the desired visualization (Figure 1). An interactive preview of the manipulations' effects, on a small-scale representation of the system, considerably tightens the customization feedback loop.

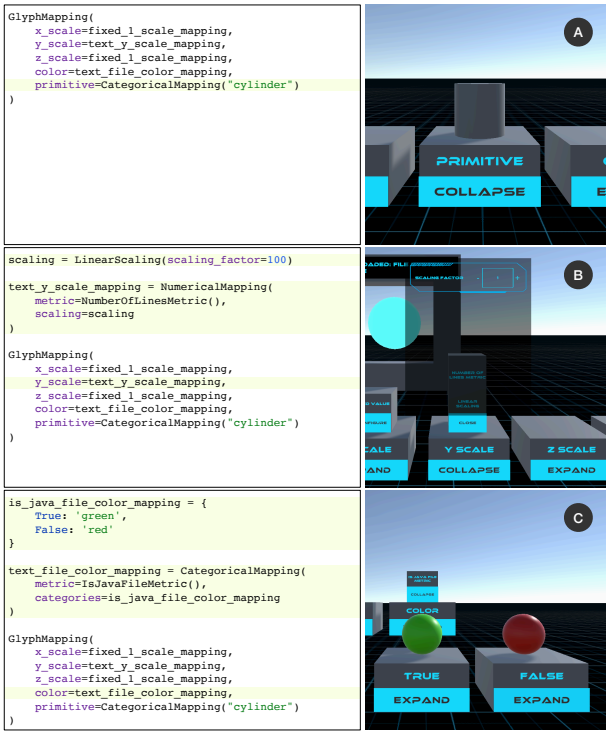

Fig. 1. Examples of 3D object manipulation and corresponding DSL code. (A) Primitive shape, (B) y-scaling, (C) categorical metric color mapping.

This work is supported by the Swiss National Science Foundation (SNSF) through the project "INSTINCT" (SNF Project No. 190113).

II. Viewspecs Manipulation in VR

Traditional software visualization tools leverage GUIs [17], [18], and DSLs [16], [19] to configure different aspects of the final expected output, in order to highlight the desired insights through the visualization. These configuration means are becoming limited and obsolete with the increasing relevance of VR-assisted software engineering (*e.g.,* see [20]–[24] just as a few examples), forcing developers to either step away from the immersive experience (*e.g.,* external tools to configure the VR environment) or to forego the potential of the VR medium by "flattening" traditional 2D GUIs in VR.

Limited foveal resolution and artifacts in the stereoscopic anti-aliasing of text add to this problem, contributing to the poor readability of 2D GUIs in VR. This, in turn, increases the importance of exploring a new way of building software visualization configurations, by managing the same primitives that represent the components of the analyzed system, while meta-entities act as manipulation proxies for the corresponding model abstractions, as shown in Figure 1.

Viewspec Configuration: The configuration process is divided in two steps. The first step (Figure 2 top) configures the three main components at system-level:

- Layout: How are the entities in the system placed in the available space (*e.g.,* linear, grid, rectangle packing);
- Scaling: The scaling to apply to all entities in the system (no scaling, fixed scaling, room-size);
- Sorting: For ordered layouts (*e.g.,* linear), the order in which the entities are considered for positioning.

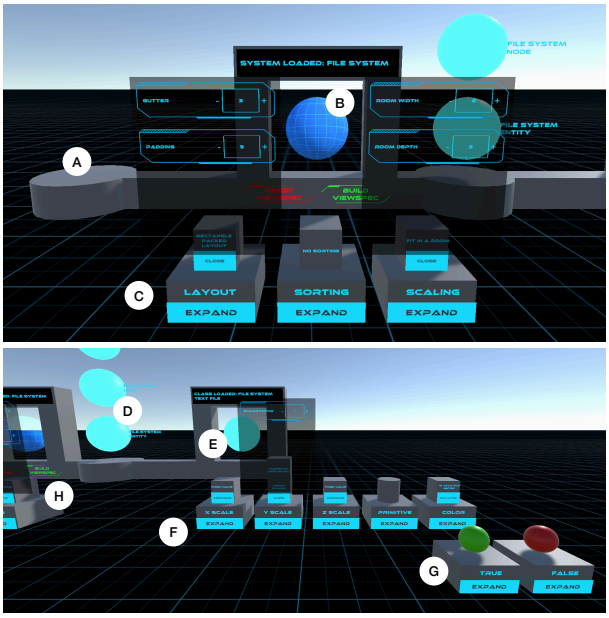

Fig. 2. Viewspec configuration interface.

After selecting a system by placing an entity from (A) in (B) and configuring system-wide parameters (C), the second step (Figure 2 bottom) configures the representation of each entity of interest in the system (D).

Each entity is selected by placing it in the selection window (E), the possible metrics and scalings can be shown for each visual attribute one at a time (F) (x/y/z-scaling, shape, color). Each metric object and an optional scaling can be placed on top of the corresponding visual attribute on which to project the (scaled) metric. For categoric metrics, each possible value can be mapped to the different values of a corresponding categoric visual property. For example, the two values of the boolean metric of being a Java file can be mapped to two different colors, green for true and red for false (G).

This process is repeated for all the entities of interest, until the configuration is completed (H). For optimization reasons the preview is updated when the mapping for an entity is complete (*i.e.,* the object sphere is removed from the selection window, see Sections V and VII for a discussion of limitations and possible improvements).

In Figure 3, we show how each saved viewspec (A) can be applied to instances of systems (B). We produce a preview of the system visualization (C) by loading (*i.e.,* placing) a system onto a viewspec.

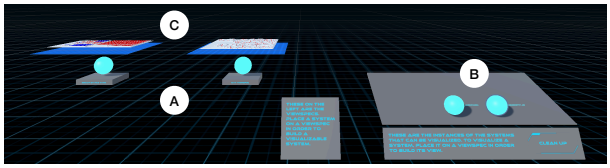

Fig. 3. Viewspec application to system instances and previews.

In Figure 4, a composable preview (C) can be created from individual systems' previews (*e.g.,* Argo UML (A) and Serenity JS (B) GitHub repositories) and acts as a proxy for the scene to be manipulated before transitioning from the configuration lobby to the actual system exploration area.

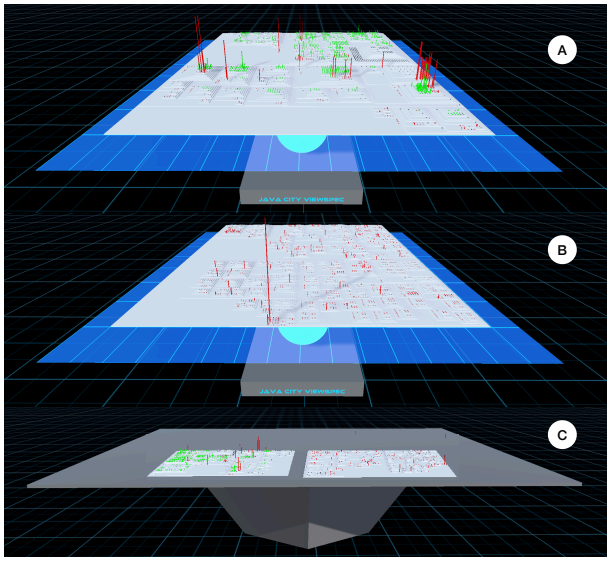

Fig. 4. Composable previews: Argo UML GitHub repository (A), Serenity JS GitHub repository (B), and composed view scene preview (C).

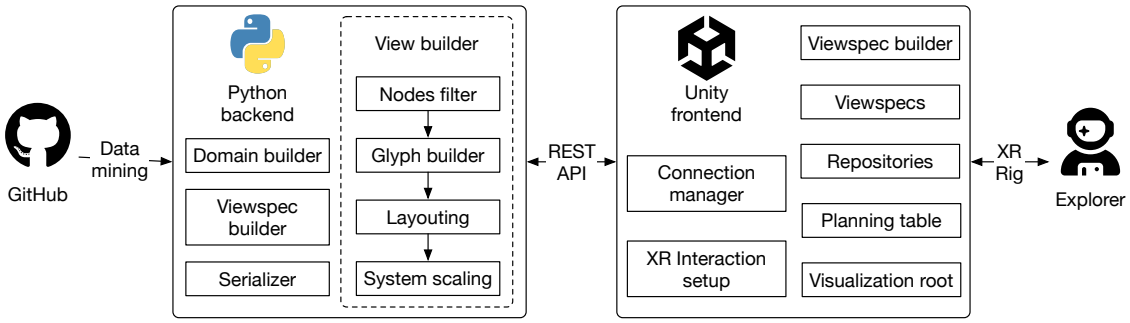

Fig. 5. Software architecture overview.

III. IVAR-NI

We developed IVAR-NI, a prototype to explore which features could make Interactive Virtual Reality-Native Interfaces usable for configuring VR software visualization applications. Figure 5 shows an overview of the software architecture.

IVAR-NI is composed of a Python backend that reifies the domain model of interest, providing the objects (*e.g.,* domain entities, viewspecs, metrics) via REpresentational State Transfer (REST) endpoints. The frontend is a Unity[1] application optimized for the Oculus Meta Quest 3[2] Head Mounted Display (HMD). The interactions in VR are carried over with two Quest 3 Touch Plus controllers (one in each hand).

The backend consists of 4 components. The *domain builder* mines a GitHub repository to build a graph of the model to be visualized. Nodes correspond to the entities (*e.g.,* folders, binary files, text files, file types) and edges to their relationships (*e.g.,* containment edges between folders and files). The *viewspec builder* uses reflection to analyze the domain and compute metrics to provide the frontend with the data to build the viewspec configuration interface. The *view builder* uses a viewspec to build the view of a domain, filtering the nodes to be displayed, building the glyphs for each node, scaling and arranging them. Finally, a *serializer* enables object transfer and communication between the backend and the frontend.

The frontend is a Unity application installed in the Meta Quest 3. The *connection manager* provides the endpoints to connect to the backend, serialize, and deserialize data. The connection parameters (*e.g.,* IP address, port) can be configured through a menu in the Head Up Display (HUD).[3] The *viewspec builder*, the *repositories*, the *viewspecs* and their composition via the *planning table* implement the customization approach described in Section II. The *visualization root* is the entry point for system exploration and is responsible for spawning and handling the glyphs. Finally, the *eXtended Reality (XR) interaction setup* collects all the components that manage user interaction with the 3D scene (*e.g.,* poke interactor, ray interactor). In particular, the XR rig, shown in Figure 6, manages input data from the controllers and the HMD and output data to the HUD and the inspection tool.

[1]See **`https://unity.com`**

[2]See **`https://www.meta.com/quest/quest-3/`**

[3]Some elements can be anchored to the HUD and displayed in a position relative to the user's point of view (*e.g.,* connection parameters menu).

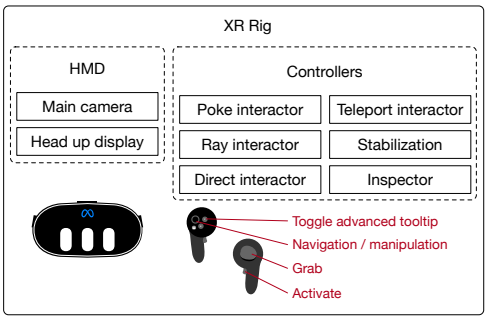

Fig. 6. XR Rig.

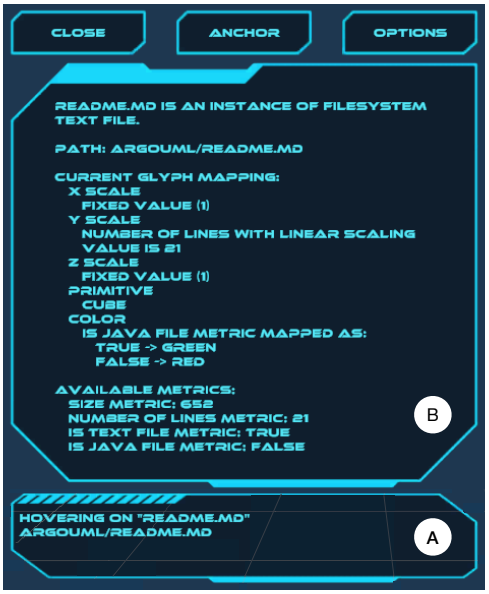

Fig. 7. Inspection tool.

The XR Rig provides custom mappings for VR *navigation* by moving in the real world (*e.g.,* walking, crouching, turning and tilting head) or through the controllers: Via teleportation (pushing the thumb-stick upward and pointing the casted ray to the desired destination) and by snap turning (pushing the thumb-stick left or right). Continuous movement and turn can be enabled[4] on the thumb-sticks of the controllers, also allowing the user to fly for a bird's eye view of the system.

[4]This control mode caused motion sickness to some users, default disabled.

Both controllers allow object *manipulation* through the grab interaction or from a distance with the casted selection ray. Grabbed objects can be rotated with the thumb-stick. Controllers' stabilization and "magnetic" rays casted from the controllers enable precise interactions even at long distances.

The *inspection* tool, shown in Figure 7, completes the toolset for system exploration: A basic tooltip (A) with concise information and an advanced tooltip (B), that can be toggled, with detailed information about the focused domain object.

IV. Discussion

We presented an approach to software visualization customization based on VR-native interfaces to avoid unnecessary context switches. The configuration steps leverage 3D shapes that match those in the software visualization (thus increasing memory recall) to represent entities and meta-entities of the explored system. The reflection-based collection of domain entities and their metrics renders this approach to configuration (and the frontend) domain-independent.

The interactivity provided by scaled-down previews of the visualization and the tuning of visualization elements directly in VR help in keeping the focus on the configuration task. The amount of usable space provided by an immersive 360 degree solution allows to experiment with scaling of the configuration options while keeping the interface functional. Some aspects of the hierarchical organization of UI components still need to be improved, to limit scene clutter and information overload.

Regarding the interactions, ambidextrous object manipulation is superior to mouse-based 2D interaction in allowing complex coordinated gestures (*e.g.,* moving controllers away from each other to scale an object in different dimensions). The number of buttons available on each controller is superior to those of a typical mouse but lacks the functionalities of a keyboard. Still, the additional degrees of freedom for object manipulation proved a key factor in shaping the customization interface in 3D. Horizontal alignment of options for visual attributes and displacement on the depth axis for the selection of each option contribute to easily controllable interactions with the interface, even at a distance (*i.e.,* with casted rays).

The scale of the 3D interface determines the effectiveness of the interactions and should be fine-tuned also with respect to the physical characteristics of the user (*e.g.,* arm reach). Our preliminary results show a promising naturalness even at significant tilt angles and distances (*e.g.,* about 45 degrees from the front of the object, 1.5 meters away). This is particularly relevant for the number of configuration options and alternatives that can be made available in a VR-native user interface. The performance comparison with different layouts and shortcut-based interfaces still needs to be evaluated.

V. Generalizability and Limitations

The reflection-based approach we propose can generalize to different domains, from a file-system, to a GitHub repository with source code files, representing entities such as classes, methods, and attributes. Integrating object customization directly in the visualization allows for a domain-independent approach to define the visual characteristics of the represented model entities. Nevertheless, we still need to validate the applicability of our approach to scale to source code granularity.

We encountered some limitations in the reliability of the plugins for the Unity frontend. The quality of the developers toolchain for VR frameworks influences the developer's experience, and these software tools are still not mature enough to be stable across different hardware configurations.

Performance-related issues are a limiting factor for the usability of the tool. The interactive preview is useful only as long as its update is fast enough to keep the configuration manipulation process smooth. Our current implementation is usable, but a more fine-grained updating strategy of the frontend components could significantly increase its performance.

To mitigate the limited internal validity of our design decisions for the prototype, during development, we utilized examples of real systems to assess the feasibility of our approach and the usability of the tool on non-synthetic data. A complete evaluation is planned as future work (Section VII).

There are also limitations in how repetitive operations need to be manually performed multiple times. To mitigate this, IVAR-NI supports the definition of default viewspecs that can be fine-tuned, minimizing the number of required operations.

VI. Related Work

VR-assisted software development is becoming the norm rather than the exception [20]–[25], yet, most of the proposed VR software visualization solutions do not allow users to build their own view specifications interactively. Merino *et al.* experimented with situated data visualizations in augmented reality, finding that live feedback is critical for user engagement [26].

The advantages of VR over non-VR visualizations are still debated [8]. Moreno *et al.* developed BabiaXR, a tool for experimenting with data visualization in XR [10]. Many tools leverage custom VR visualizations for collaborative [12] software architecture analysis and recovery [9], [11], [13], exploration of large software systems [7], and dependency ecosystems analysis [14]. While the software visualization part of these works provides significant contributions, the configurability of the tools is often limited to standard metaphors, slightly adapted to the VR context, that do not feel natural.

LaViola *et al.* [27] provide guidelines on design of 3D and VR user interfaces, highlighting how virtual environments can provide the user with a sense of presence. By designing natural interactions through lifelike metaphors, one can take advantage of the knowledge the user already has of the real world, thus reducing the cognitive distance between real and virtual [27].

Lanza presented guidelines on configuration of traditional visualization tools [17] learned while developing CodeCrawler [18]. Its view editor window is a GUI with selection lists and drop-down menus. Similarly, Raglianti *et al.* used viewspecs with 2D software visualization tools for developers communications on Discord [19], [28]. These viewspecs are configurable in a simple internal DSL (using the Pharo language syntax) and we used them as a starting point for the system-wide configuration (see Section II).

VII. Future Work

Technical Improvements: IVAR-NI is still a stand-alone proof-of-concept. We plan to explore its integration as a library to enhance other projects. We will also improve both the backend and the frontend. Better support for automatic parsing of the system's architecture will allow, through the same reflection mechanisms we already use, to capture more fine-grained aspects of the system and further tune its visualization. Increasing the responsiveness of the preview is also crucial.

New Features: Our current approach is limited to customizing the visualization of the system, but the whole explorable scene could be configured "from inside". We will expand the *planning table* component to integrate the definition of parameters in a preview lobby, without leaving the VR environment, for an immersive and configurable end-to-end experience.

User Study Evaluation: We will perform a complete evaluation with a pilot study and a larger user study. The former will be targeted at tweaking design decisions of the prototype, the latter will gather feedback through surveys and semi-structured interviews. We aim to assess the impact of eliminating context switches that bring the user away from a fully immersive environment and interaction metaphor.

VIII. Conclusion

Rethinking configuration of software visualization in VR can be highly beneficial. The trend to explore VR solutions to assist software engineers and software architect, coupled with the maturity level of modern VR headsets opens up novel and interesting paths. The proposed approach is just a first step in the direction of VR-native user interfaces for software visualization customization. Nevertheless, keeping software developers and engineers immersed in the system visualization and giving them interactive means to affect it from within, while using natural metaphors coherent with the VR environment surrounding them, is key to keeping them in the flow and reducing unnecessary context switches.

Demo video: `https://youtu.be/HsWGtrINtHc`

References

[1] A. Theodoropoulos and A. Antoniou, "VR games in cultural heritage: A systematic review of the emerging fields of virtual reality and culture games," *Applied Sciences*, vol. 12, no. 17, p. 8476, 2022.

[2] C. Wang, Y. Tang, M. A. Kassem, H. Li, and B. Hua, "Application of VR technology in civil engineering education," *Computer Applications in Engineering Education*, vol. 30, no. 2, pp. 335–348, 2022.

[3] L. Li, F. Yu, D. Shi, J. Shi, Z. Tian, J. Yang, X. Wang, and Q. Jiang, "Application of virtual reality technology in clinical medicine," *American Journal of Translational Research*, vol. 9, no. 9, p. 3867, 2017.

[4] E. H. Korkut and E. Surer, "Visualization in virtual reality: A systematic review," *Virtual Reality*, vol. 27, no. 2, pp. 1447–1480, 2023.

[5] T. D. Goddard, A. A. Brilliant, T. L. Skillman, S. Vergenz, J. Tyrwhitt-Drake, E. C. Meng, and T. E. Ferrin, "Molecular visualization on the holodeck," *Journal of Molecular Biology*, vol. 430, no. 21, pp. 3982–3996, 2018.

[6] S. Pradhan, V. Nanniyur, and P. K. Vissapragada, "On the defect prediction for large scale software systems — From defect density to machine learning," in *Proceedings of QRS 2020 (International Conference on Software Quality, Reliability and Security)*, 2020, pp. 374–381.

[7] A. Hoff, L. Gerling, and C. Seidl, "Utilizing software architecture recovery to explore large-scale software systems in virtual reality," in *Proceedings of VISSOFT 2022 (Working Conference on Software Visualization)*. IEEE, 2022, pp. 119–130.

[8] D. Moreno-Lumbreras, G. Robles, D. Izquierdo-Cortázar, and J. M. González-Barahona, "Software development metrics: To VR or not to VR," *Empirical Software Engineering*, vol. 29, no. 2, p. 42, 2024.

[9] A. Hoff, C. Seidl, and M. Lanza, "Uniquifying architecture visualization through variable 3D model generation," in *Proceedings of VaMoS 2023 (International Working Conference on Variability Modelling of Software-Intensive Systems)*. ACM, 2023, pp. 77–81.

[10] D. Moreno-Lumbreras, J. M. González-Barahona, and G. Robles, "BabiaXR: Facilitating experiments about XR data visualization," *SoftwareX*, vol. 24, p. 101587, 2023.

[11] 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 ESEC/FSE 2021 (Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering)*. ACM, 2021, pp. 1455–1458.

[12] A. Hoff, M. Lungu, C. Seidl, and M. Lanza, "Collaborative software exploration with multimedia note taking in virtual reality," in *Proceedings of ICPC 2024 (International Conference on Program Comprehension)*. ACM, 2024, pp. 346–357.

[13] ——, "Immersive software archaeology: Collaborative exploration and note taking in virtual reality," in *Proceedings of ICPC 2024 (International Conf. on Program Comprehension)*. ACM, 2024, pp. 387–391.

[14] D. Moreno-Lumbreras, J. M. González-Barahona, and M. Lanza, "Understanding the NPM dependencies ecosystem of a project using virtual reality," in *Proceedings of VISSOFT 2023 (Working Conference on Software Visualization)*. IEEE, 2023, pp. 84–94.

[15] D. Moreno-Lumbreras, J. M. González-Barahona, G. Robles, and V. Cosentino, "The influence of the city metaphor and its derivates in software visualization," *Journal of Systems and Software*, vol. 210, p. 111985, 2024.

[16] N. Rentz and R. von Hanxleden, "SPViz: A DSL-driven approach for software project visualization tooling," *arXiv preprint*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.17063

[17] M. Lanza, "CodeCrawler—Lessons learned in building a software visualization tool," in *Proceedings of CSMR 2003 (European Conference on Software Maintenance and Reengineering)*. IEEE, 2003, pp. 409–418.

[18] M. Lanza, S. Ducasse, H. Gall, and M. Pinzger, "CodeCrawler: An information visualization tool for program comprehension," in *Proceedings of ICSE 2005 (International Conference on Software Engineering)*. ACM, 2005, pp. 672–673.

[19] M. Raglianti, C. Nagy, R. Minelli, and M. Lanza, "DiscOrDance: Visualizing software developers communities on Discord," in *Proceedings of ICSME 2022 (International Conference on Software Maintenance and Evolution)*. IEEE, 2022, pp. 474–478.

[20] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "CityVR: Gameful software visualization," in *Proceedings of ICSME 2017 (Int. Conf. on Software Maintenance and Evolution)*. IEEE, 2017, pp. 633–637.

[21] J. Vincur, P. Navrat, and I. Polasek, "VR City: Software analysis in virtual reality environment," in *Proceedings of QRS-C 2017 (International Conference on Software Quality, Reliability and Security Companion)*. IEEE, 2017, pp. 509–516.

[22] A. Hori, M. Kawakami, and M. Ichii, "CodeHouse: VR code visualization tool," in *Proceedings of VISSOFT 2019 (Working Conference on Software Visualization)*. IEEE, 2019, pp. 83–87.

[23] S. Romano, N. Capece, U. Erra, G. Scanniello, and M. Lanza, "On the use of virtual reality in software visualization: The case of the city metaphor," *Information and Software Techn.*, vol. 114, pp. 92–106, 2019.

[24] M. Inkarbekov, R. Monahan, and B. A. Pearlmutter, "Visualization of AI systems in virtual reality: A comprehensive review," *arXiv preprint*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.15545

[25] R. Oberhauser and C. Lecon, "Virtual reality flythrough of program code structures," in *Proceedings of VRIC 2017 (Virtual Reality International Conference)*. ACM, 2017, pp. 1–4.

[26] L. Merino, B. Sotomayor-Gómez, X. Yu, R. Salgado, A. Bergel, M. Sedlmair, and D. Weiskopf, "Toward agile situated visualization: An exploratory user study," in *Proceedings of CHI 2020 Extended Abstracts (Conference on Human Factors in Computing Systems, Extended Abstracts)*. ACM, 2020, pp. 1–7.

[27] J. J. LaViola Jr., E. Kruijff, R. P. McMahan, D. A. Bowman, and I. Poupyrev, *3D User Interfaces: Theory and Practice*. Addison-Wesley Professional, 2017.

[28] M. Raglianti, R. Minelli, C. Nagy, and M. Lanza, "Visualizing Discord servers," in *Proceedings of VISSOFT 2021 (Working Conference on Software Visualization)*. IEEE, 2021, pp. 150–154.