# Code Refactoring in Virtual Reality

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

***Abstract*—Refactoring source code is a key technique for maintaining a high-quality codebase, keeping the code clean, modular, understandable, and adaptable to change in the long run. Modern Integrated Development Environments (IDEs), unlike plain text editors, provide automated support for various refactorings (*e.g.,* move class, extract method). IDEs have seen major advances over the decades, but remain constrained by conventional bento-box interfaces and input methods (*e.g.,* keyboard). Recent advances in Virtual Reality (VR) and eXtended Reality (XR) technology, have opened up the possibility of rethinking IDEs, where the I does not stand for integrated, but for immersive.**

**We present a novel approach for refactoring source code in VR, combining customizable software visualizations and the interaction capabilities of modern VR controllers. Whereas existing research on depicting software in VR has remained within the realm of "read-only" comprehension, we take it a step further, making it possible to rewrite the underlying codebase by performing interactions in VR which encode refactorings. We present two examples where, through ambidextrous interactions and controller triggers, developers can gather information about the system and modify it at different abstraction levels. We conclude with a reflection on the integration of XR features into modern IDEs versus the development of new standalone XR-native IDEs.**

***Index Terms*—Refactoring, Virtual Reality, VR Interaction, Software Visualization**

## I. Introduction

As software systems grow in complexity over time, maintaining a high-quality codebase becomes an ongoing challenge [1]. Changes in requirements, addition of new features, and rapid iteration cycles often lead to degraded architectures [2]–[4], bloated and tangled source code [5], making maintenance difficult and reducing the overall adaptability of the system [1], [5]. Refactoring is a critical activity for addressing these issues [6], [7]. By taking the time to refactor, developers can simplify the code, making it easier for themselves and others to navigate. This clarity reduces the cognitive load on developers, allowing them to focus on problem-solving rather than deciphering tangled code [8]. Refactoring helps developers ensure that the system remains clean, modular, and easier to work with [1].

Despite their importance, refactorings are often hindered by the limitations of current Integrated Development Environments (IDEs), like difficulties in feature discoverability [9] and lack of trust in the correctness of IDE refactorings [10]. Developers prefer manual refactorings [10], missing the point that they are supposed to be *automated* code transformations [11].

Whether in the IDE or in a traditional text editor, when it comes to refactorings, developers prefer find-and-replace strategies, relying on two-dimensional text-based interfaces and conventional input methods (*e.g.,* mouse, keyboard).

This consolidated ecosystem struggles to break away from known metaphors and technologies, leaving developers with sub-optimal tools [12] to understand and maintain increasingly complex and interconnected software systems.

We propose an approach to refactoring based on Virtual Reality (VR) and leverage the (6 degrees of) freedom of modern VR controllers to interact with source code at different levels of abstraction (see, for example, Figure 1).

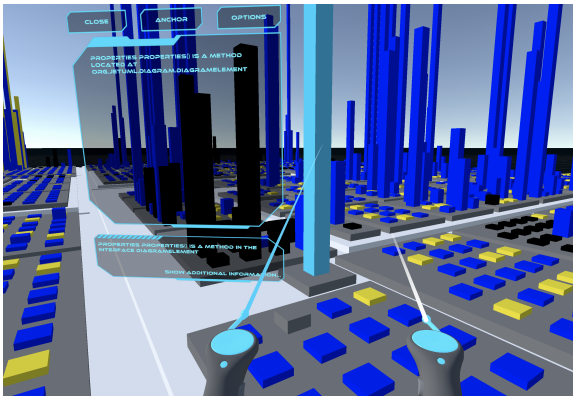


Fig. 1. Example visualization and manipulation of source code in VR.

The immersive three-dimensional environment provides an intuitive visualization of the codebase, effectively guiding the refactoring operations. The support of Language Servers[1] provides an abstraction layer to the implementation of automatic refactorings. We show how to map the actions available in VR to the corresponding sequence defining a refactoring.

Our approach combines customizable visualizations with rich interaction mechanisms, exploring VR software engineering beyond program comprehension, with practical uses for *active* software maintenance and evolution. We aim to improve how developers perceive the refactoring activity and simplify it by making the process more organic, engaging, and less burdened by text-related technicalities. The *move class* and *extract method* case studies show a paradigm shift from a 2D textual workspace to a dynamic 3D environment with abstract (yet informative) representations of code elements. The developer becomes spatially aware of the code and its relationships, based on the context and at multiple levels.

We conclude with a broader reflection, encompassing eXtended Reality (XR), on advantages and disadvantages of two contrasting approaches: Bringing XR in current IDEs versus developing a new concept of XR-native IDE, exploring new metaphors and leveraging new powerful interactions.

[1]See `https://github.com/eclipse-jdtls/eclipse.jdt.ls`

## II. Visualization and Interactions

Our approach combines customizable software visualization with the interaction capabilities of VR to create a more intuitive and immersive experience for developers. Traditional code representations are replaced with VR visualizations that enable developers to explore, manipulate, and understand their codebase in 3D.

We represent code elements such as classes, methods, and dependencies as 3D objects (*i.e.,* glyphs) in a Virtual Environment (VE). The glyphs are arranged to reflect logical groupings and relationships, enabling developers to quickly grasp the overall structure of the system. For example, a codebase can be visualized through the city metaphor [13], as shown in Figure 2. Classes are represented as buildings, laid out by a rectangle packing algorithm, and packages as neighborhoods. The visualization can be enriched by depicting dependencies between entities as connecting lines.

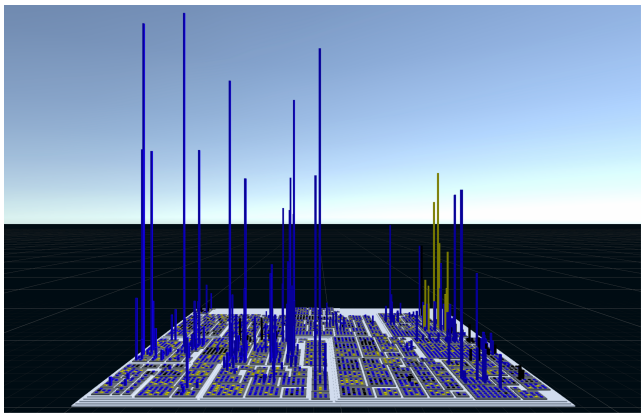

Fig. 2. Visualizing JetUML through the city metaphor.

Two hand-held controllers allow interaction with the VE, either with nearby objects by direct contact or with distant ones using a ray cast from the controller. Besides grabbing items, controller buttons enable developers to access additional features, such as visualizing the dependencies between components. Buttons can be used during other interactions without disrupting the workflow. Some interactions support progressive disclosure of information: For example, the thumb-stick can be used to activate the inspection tool, showing additional information on-demand and at different levels of detail.

The user can interact with a glyph by grabbing and moving it around the VE or by casting a ray from a distance. Moving a glyph has its own semantic. For example, positioning an element inside a hierarchy can correspond to moving a file to a location in a nested file system path. Similarly, moving implies constraints, naturally guided, enforced, and augmented by visual feedback. For example, if the glyph is released in an invalid position, it will be colored in red and automatically return to its place upon release. If it is released in a legal position, a preview is shown. On release, the corresponding domain entity is moved accordingly.

The large number of available combinations to define complex interaction sequences led us to model them using Finite State Automata (FSA), such as the one shown in Figure 3.

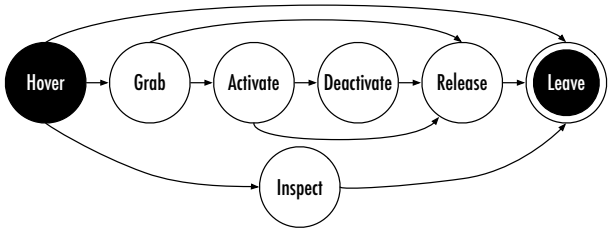


Fig. 3. FSA describing a generic interaction.

Each state of the FSA represents a VR interaction that the user can perform on any glyph:

- **Hover**: The ray cast from the controller is placed on the glyph. The glyph is highlighted and an haptic feedback is reproduced on the controller.
- **Leave**: The ray cast from the controller is moved away from the glyph, which returns to its original unhighlighted status. An haptic feedback is reproduced on the controller.
- **Activate**: Button-based interaction. A domain-specific action can be triggered by this interaction.
- **Deactivate**: Button-based interaction. A domain-specific action can be triggered by this interaction.
- **Grab**: The glyph is grabbed and moved in the VE.
- **Release**: The glyph is released in the current position. The interaction can have a semantic to check if the position is valid, otherwise the glyph is placed back in the original position. If a glyph is active, it is also automatically deactivated when released.
- **Inspect**: Button-based interaction. Can be used to present additional information.

Using two VR controllers, developers can perform multiple complex interactions at the same time. For instance, moving a class from one module to another can be accomplished by grabbing the class and placing it in the desired location. Meanwhile, the other controller can be used to collect information about other classes or the intended destination package. Moreover, traveling into the VE is carried out by walking into the physical environment, hence codebase navigation can be performed simultaneously with controller interactions.

## III. Refactoring Examples — Two Case Studies

We illustrate our approach presenting two refactoring case studies: *move class* and *extract method*. In the first one, a developer needs to relocate a Java class from one package to another. In a traditional IDE, this task would require navigating multiple files to understand dependencies, manually identifying the best target package, and executing the refactoring through a contextual menu on the class file, followed by the selection of the destination package. In Visual Studio Code, a popular multi-language IDE, the destination package would be selected by browsing through a lexicographically sorted list of all the possible destination packages. We implemented this refactoring in a tool which we present in section IV.

In the second case study, a developer needs to break down a large method into smaller ones, extracting parts of the logic in one or more separate functional units (*i.e.,* other methods). This refactoring improves code readability by increasing abstraction levels and helps reducing code duplication. We present this case study to show a lower-level refactoring, directly manipulating lines of code.

The developer begins by entering the VE, where the codebase is visualized through the city metaphor. Classes are depicted as buildings and packages are depicted as districts. Following the city metaphor, the size of the building represents metrics of the class. The number of methods and fields of the class are mapped, respectively, on the length and width of the building's base, while the number of lines of code is mapped on the height of the building.

The number of references to the class is mapped on the color saturation of the building (see Figure 4). A low-saturation red represents a low number of references, a high-saturation red vice-versa. Reference destinations are represented by lines connecting the components of the codebase. Given the potentially high number of references, they are hidden and visualized only on-demand, to avoid unnecessary cluttering.

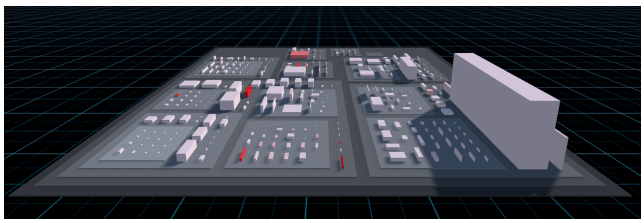

Fig. 4. Example visualization.

### A. Move Class Refactoring in VR

In Figure 5, we show part of the sequence depicting the interactions to perform the move class refactoring. The class to be moved is highlighted by its color (A), and the developer can explore this region of the city to understand the context of the operation to be performed. To initiate the refactoring, the developer places the ray on the class (A) and then grabs it using one controller, dragging it to the desired package (B). The backend performs the refactoring and dynamically updates the visualization to reflect the change, allowing the developer to assess the impact of the refactoring in real-time.

While moving the class, the developer can use the other controller to perform additional interactions, like collecting information about other entities (C), to inform the choice of the destination package. Similarly, the trigger button can be used to visualize the dependencies (D) for choosing the package that minimizes the number of external references.

By inspecting the meta-data and visualizing metrics associated with the class, the developer can also evaluate the current status of the codebase. This seamless interactive process contrasts sharply with the fragmented and "under-informed" workflow of traditional IDE refactorings, highlighting the advantages of our approach.

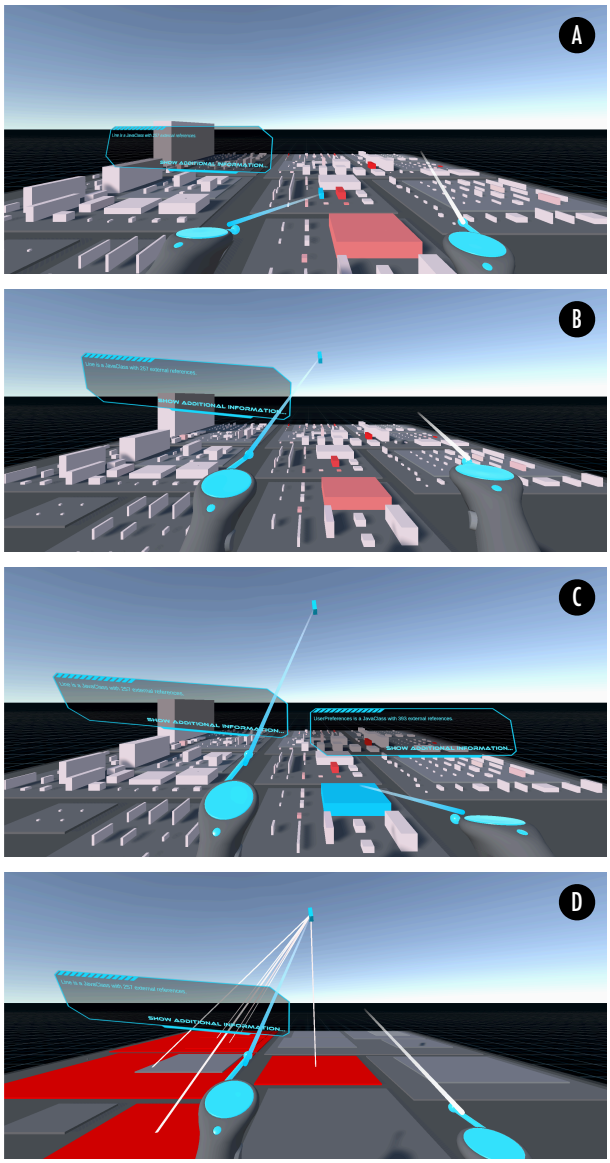


Fig. 5. To perform a move class refactoring, the developer: (A) chooses the class to move, (B) grabs it and lifts it, (C) optionally inspects other elements and (D) visualizes dependencies to better understand the context.

### B. Extract Method Refactoring in VR

A possible implementation of this refactoring in VR requires a more fine-grained representation of the source code, since it involves visualizing and interacting with individual statements in the source code. Each statement in the method could be represented as a 3D block, enabling developers to group, select, manipulate, and restructure them using the controllers.

When a developer needs to extract a portion of a method, they use one controller to point to the first instruction of the block they want to refactor and press a button to activate the instruction, thus anchoring the selection. As they drag the controller down to the last instruction, the active instructions between the first and last one are progressively highlighted, and visually connected to indicate the active grouping. A common handle allows moving the selected block as a unit.

Once the selection is complete, the developer can easily extract the block by *grabbing* the handle and *dragging* it out of the original method. The system automatically creates a new method and a corresponding glyph representing it. The selected instructions become the body of the new method, and the tool automatically inserts a call to the new method in place of the original code block. The process can be enhanced by highlighting the variables from the original method that are used within the extracted code. This feature allows the developer to visualize variable dependencies and manage them with the other controller before finalizing the refactoring, ensuring that the behavior of the original code is preserved.

## IV. Implementation: Challenges and Limitations

We developed a tool to explore our approach and tested it on a Meta Quest 3 headset. We implemented a Python-based backend and a frontend in Unity, communicating through a REST API for data exchange. The backend is responsible for reifying the codebase into a graph model, while the frontend renders the VE and manages user interaction. An overview of the architecture of the tool is shown in Figure 6.

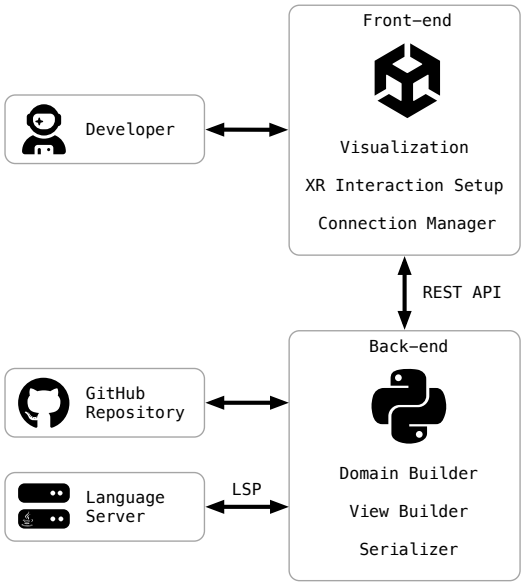


Fig. 6. Software architecture of the tool.

The **backend** consists of three main components. The *Domain Builder* scans the filesystem of a repository to create a tree structure where nodes represent files and directories, and edges represent their containment relationships. The Domain Builder also interfaces with a Language Server (LS) to parse source code files, extracting language entities (*e.g.,* classes, methods) to integrate them into the graph model. The *View Builder* maps software metrics (*e.g.,* lines of code) to visual attributes (*e.g.,* glyph height) and positions the glyphs according to a chosen layout (*e.g.,* rectangle packing).

Users are able to modify mappings and layouts as described in our previous work [14], customizing the visual representation of source code elements. The *Serializer* handles the data to be served to the frontend through the REST API.

The main challenge we faced during backend development was model instantiation. The LS is a service that supports IDEs with textual information, usually designed to be printed on screen. To address this, we developed a middleware that implements the Language Server Protocol (LSP)[2] and handles the communication between the backend and the LS, thus opening the possibility to support different programming languages.

The **frontend** consists of three main components. The *Connection Manager* fetches the data from the backend and prepares them for the visualization. The *Visualization* component is responsible for spawning and handling the 3D glyphs corresponding to elements of the codebase. The *XR Interaction Setup* manages the interactions with the developer, reading data from the headset and the controllers (*e.g.,* position, rotation) to update the VE accordingly. It also allows the user to move in the VE and to manipulate the virtual objects.

Frontend challenges concern interaction and immersion. Interaction design draws from human-computer interaction principles, such as affordances, feedback, and ergonomics, to ensure intuitive user experiences. Immersion, a measurable aspect of VR, also depends on factors like display resolution and refresh rate, and can be affected by the limitations of VR hardware. While SDKs offer tools for interaction and performance optimization, they often introduce compatibility issues, leading to development environment instability.

The use of VR presents both accessibility and usability challenges. Many developers are unfamiliar with VR technologies and the learning curve for mastering a new paradigm may be a deterrent. Additionally, the cost of VR hardware and the need for dedicated setups can limit its adoption. Developers who are accustomed to the efficiency of keyboard shortcuts and textual interfaces may find the immersive VR workflow less effective. Furthermore, the lack of a keyboard for text input in VR remains a limitation, as the controllers, although more capable in a variety of situations, are not ideal for text input.

Scalability is another challenge. Visualizing large codebases in 3D can lead to cluttered and overwhelming views, making it difficult for users to focus on specific elements. This is why we integrated features to manage visual complexity (*e.g.,* entity filtering, different layouts), nevertheless, we are still exploring and improving them. Finally, performance-related issues also contribute to the applicability to large-scale codebases, especially on mid-range devices, such as the Meta Quest 3 on which we developed and tested our tool.

## V. Related Work

Developers spend about 70% of their time understanding programs [15]. Software visualization is a means to ease the task [16]. Graphical representations of aspects such as code architecture [13], [17] and execution flow [18], [19] are crucial for managing complex systems [16]. Von Mayrhauser and Vans emphasized the role of program understanding in software maintenance and evolution, leveraging existing knowledge to gain new insights [8].

[2]See `https://microsoft.github.io/language-server-protocol/`

Different tools use different metaphors to represent reality [13], or focus on specific targets, such as object-oriented software [20] and dynamic execution trace analysis [21]. Hoff *et al.* developed ISA-VR, a collaborative tool to analyze the activity of distributed teams within VEs [22].

VR has evolved from early stereoscopic photography to modern headsets like the Meta Quest and the HTC Vive [23]. Initially constrained by high costs, VR is now widely used in fields such as healthcare [24], training [25], and software engineering [14], [26]–[28]. The interaction capabilities of VR make it particularly suited for software visualization, enabling developers to explore systems in 3D, improving comprehension, and fostering collaboration [22].

Despite the popularity of VR-assisted software engineering among researchers for its visualization, comprehension, and collaboration capabilities [22], [26], [29]–[33], little research is carried on about interactions with software elements in VR. Input methods, including controllers, hand gestures, voice commands, and eye tracking are essential for usability and immersion [34] as well as accessibility to enhance user experience [35]–[37]. Developing VR systems for software visualization requires balancing high-performance rendering, user comfort, and design to leverage the full potential of XR technologies in software development [38].

## VI. XR in the IDE vs. IDE in XR

The following reflection encompasses the integration of XR technologies in general, both VR and Augmented Reality (AR), into software development workflows. XR presents opportunities to rethink how developers interact with their tools. Two conceptual approaches emerge: Embedding XR capabilities within traditional IDEs (*i.e.,* XR in the IDE) and creating fully immersive development environments where the IDE exists entirely in XR (*i.e.,* IDE in XR).

In the "XR in the IDE" model, XR features are added, for example via AR visualization plugins, to enhance existing IDEs. This approach retains the familiarity of conventional IDEs, allowing developers to adopt XR gradually, to augment current capabilities, without abandoning established workflows. For instance, an IDE could provide an XR mode where developers visualize dependencies in 3D or analyze large-scale software architectures. However, this approach often constrains XR's potential, as it forces context switches breaking the flow of immersive experiences. The approach is also limited by the quality of current XR hardware, especially for VR, forcing the user to wear and remove the headset when going from the IDE to the VR experience.

Conversely, the "IDE in XR" paradigm, the one favored in our approach, fully embraces the capabilities of XR by building an immersive development environment designed specifically for XR interactions. This approach redefines how developers interact with their code by leveraging spatial representations and natural input mechanisms, such as two-handed source code manipulation with a greater freedom to design innovative workflow interactions.

We visualize code structures as interactive glyphs, allowing developers to directly manipulate elements based on the knowledge acquired through exploration of complex relationships in the codebase, without ever leaving the XR environment or switching back to traditional input methods.

We showed how the "IDE in XR" approach can reshape software engineering workflows, using refactoring tasks to showcase a more intuitive way to design and leverage interaction sequences in VR. At the same time, this proof-of-concept highlights the importance of balancing innovation with usability, ensuring that the tools remain practical and effective for developers. The concerted evolution of this approach to the point where it can match or surpass current development lifecycles remains to be proven.

## VII. Future Work

**Technical Improvements:** A limitation of our prototype is the time to build a complete scene, which hinders real-time interaction. Optimizing this process is needed to support smoother and more dynamic interactions, tightening the feedback loop for exploration. Also, enriching the model with more fine-grained data from the LS will enhance the possibilities to define meaningful interactions for other types of refactorings. We also aim to implement the approach as an IDE plugin, evaluating the unexplored "XR in the IDE" scenario and highlighting benefits and limitations of the approach.

**New Features:** Expanding the system's ability to support more refactorings will enhance the tool's utility in practical use cases, beyond this proof-of-concept. We plan to experiment with other interaction techniques, such as hand-tracking to allow direct manipulation of virtual objects.

**User Study Evaluation:** A large-scale user study on a real-world refactoring task will allow a thorough evaluation of the approach in the wild. Moreover, we will use qualitative results to refine the interface and the interaction sequences, towards a full-fledged tool for automatic software refactoring in VR.

## VIII. Conclusion

We introduced a refactoring approach in VR, enabling refactoring through intuitive and powerful interactions. Our approach leverages immersive 3D visualizations to support exploration and knowledge acquisition.

We emphasize the interactivity of the process, an alternation between exploration and interaction, enabling developers to better understand their codebases while improving them. Our approach addresses the limited program comprehension capabilities of refactoring tools available in traditional IDEs. While many challenges remain, not only related to performance, our approach breaks the ground for a paradigm switch, leveraging the potential of XR interactions in software engineering.

**Acknowledgments:** This work is supported 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] M. Fowler, *Refactoring: Improving the Design of Existing Programs*. Pearson, 1999.

[2] L. de Silva and D. Balasubramaniam, "Controlling software architecture erosion: A survey," *Journal of Systems and Software*, vol. 85, no. 1, pp. 132–151, 2012.

[3] S. Herold, M. Blom, and J. Buckley, "Evidence in architecture degradation and consistency checking research: Preliminary results from a literature review," in *Proccedings of ECSAW 2016 (European Conference on Software Architecture Workshops)*. ACM, 2016, pp. 1–7.

[4] R. Li, P. Liang, M. Soliman, and P. Avgeriou, "Understanding software architecture erosion: A systematic mapping study," *Journal of Software: Evolution and Process*, vol. 34, no. 3, p. e2423, 2022.

[5] C. Politowski, F. Khomh, S. Romano, G. Scanniello, F. Petrillo, Y.-G. Guéhéneuc, and A. Maiga, "A large scale empirical study of the impact of Spaghetti Code and Blob anti-patterns on program comprehension," *Information and Software Technology*, vol. 122, p. 106278, 2020.

[6] T. Mens and T. Tourwe, "A survey of software refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[7] M. Pizka, "Straightening spaghetti-code with refactoring?" in *Proceedings of SERP 2004 (International Conference on Software Engineering Research and Practice)*, vol. 2. CSREA Press, 2004, pp. 846–852.

[8] A. Von Mayrhauser and A. M. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, pp. 44–55, 1995.

[9] Y. Golubev, Z. Kurbatova, E. A. AlOmar, T. Bryksin, and M. W. Mkaouer, "One thousand and one stories: A large-scale survey of software refactoring," in *Proceedings of ESEC/FSE 2021 (Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering)*. ACM, 2021, pp. 1303–1313.

[10] D. Silva, N. Tsantalis, and M. T. Valente, "Why we refactor? Confessions of GitHub contributors," in *Proceedings of FSE 2016 (SIGSOFT International Symposium on Foundations of Software Engineering)*. ACM, 2016, pp. 858–870.

[11] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of refactoring challenges and benefits at Microsoft," *IEEE Transactions on Software Engineering*, vol. 40, no. 7, pp. 633–649, 2014.

[12] R. Minelli, A. Mocci, R. Robbes, and M. Lanza, "Taming the IDE with fine-grained interaction data," in *Proceedings of ICPC 2016 (International Conference on Program Comprehension)*. IEEE, 2016, pp. 1–10.

[13] R. Wettel and M. Lanza, "CodeCity: 3D visualization of large-scale software," in *Proceedings of ICSE-Companion 2008 (International Conference on Software Engineering Companion)*. ACM, 2008, pp. 921–922.

[14] M. Giannaccari, M. Raglianti, and M. Lanza, "Manipulating VR-Native user interfaces for software visualization customization," in *Proceedings of VISSOFT 2024 (Working Conference on Software Visualization)*. IEEE, 2024, pp. 111–115.

[15] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer — An investigation of how developers spend their time," in *Proceedings of ICPC 2015 (International Conference on Program Comprehension)*. IEEE, 2015, pp. 25–35.

[16] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Science & Business Media, 2007.

[17] 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.

[18] D. F. Jerding, J. T. Stasko, and T. Ball, "Visualizing interactions in program executions," in *Proceedings of ICSE 1997 (International Conference on Software Engineering)*. ACM, 1997, pp. 360–370.

[19] T. D. LaToza and B. A. Myers, "Visualizing call graphs," in *Proceedings of VL/HCC (Symposium on Visual Languages and Human-Centric Computing)*. IEEE, 2011, pp. 117–124.

[20] 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.

[21] B. Cornelissen, A. Zaidman, D. Holten, L. Moonen, A. Van Deursen, and J. J. Van Wijk, "Execution trace analysis through massive sequence and circular bundle views," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2252–2268, 2008.

[22] 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.

[23] S. M. LaValle, *Virtual Reality*. Cambridge University Press, 2023.

[24] E. B. Foa, "Prolonged exposure therapy: Past, present, and future," *Depression and Anxiety*, vol. 28, no. 12, pp. 1043–1047, 2011.

[25] J. Whyte, *Virtual Reality and the Built Environment*. Routledge, 2007.

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

[27] 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.

[28] A. Hoff, C. Seidl, and M. Lanza, "Immersive software archaeology: Exploring software architecture and design in virtual reality," in *Proceedings of SANER 2024 (International Conference on Software Analysis, Evolution and Reengineering)*. IEEE, 2024, pp. 47–51.

[29] 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.

[30] 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.

[31] 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.

[32] 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 Technology*, vol. 114, pp. 92–106, 2019.

[33] M. Inkarbekov, R. Monahan, and B. A. Pearlmutter, "Visualization of AI systems in virtual reality: A comprehensive review," *arXiv:2306.15545 preprint*, pp. 1–19, 2023.

[34] G. C. Burdea and P. Coiffet, *Virtual Reality Technology*. John Wiley & Sons, 2024.

[35] J. Kreimeier, S. Hammer, D. Friedmann, P. Karg, C. Bühner, L. Bankel, and T. Götzelmann, "Evaluation of different types of haptic feedback influencing the task-based presence and performance in virtual reality," in *Proceedings of PETRA 2019 (International Conference on Pervasive Technologies Related to Assistive Environments)*. ACM, 2019, pp. 289–298.

[36] D. M. Cook, D. Dissanayake, and K. Kaur, "Virtual reality and older hands: Dexterity and accessibility in hand-held VR control," in *Proceedings of CHIuXiD 2019 (International Conference in Cooperation, HCI and UX)*. ACM, 2019, pp. 147–151.

[37] J. Dudley, L. Yin, V. Garaj, and P. O. Kristensson, "Inclusive immersion: A review of efforts to improve accessibility in virtual reality, augmented reality and the metaverse," *Virtual Reality*, vol. 27, no. 4, pp. 2989–3020, 2023.

[38] 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.