System Comprehension and Source Code Manipulation in eXtended Reality

Mattia Giannaccari
Supervised by Prof. Dr. Michele Lanza
REVEAL @ Software Institute — USI, Lugano, Switzerland
mattia.giannaccari@usi.ch

Abstract

Comprehending a codebase requires developers to construct mental models spanning multiple levels of abstraction, from architectural overviews to individual components. These components differ across paradigms (*e.g.,* classes in object-oriented systems, functions in functional programming). Software visualization supports this process by translating abstract software structures into perceptually meaningful forms that align with human cognitive processes. When grounded in familiar metaphors, such as cities or landscapes, visualization can further narrow the gap between abstract program constructs and developers' mental models.

How can language-agnostic expressive metaphors, direct manipulation, and extended-reality be integrated to create immersive, explorable virtual-city environments that help developers explore, understand, and refactor complex, large software systems?

Our research advances this idea by extending the city metaphor to represent classes and packages as buildings and districts, where architectural details encode structural and behavioral properties. We classify classes into *Class Archetypes* using heuristics based on metrics, intra-class, and inter-class relationships. We then develop a specific building for each archetype: Utility classes become factories, data classes become depots, constant definers become archives, and adapter classes appear as surrounding structures. Building on this foundation, we aim to develop a unified, interactive, and immersive visual language for program comprehension.

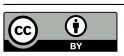
CCS Concepts

• **Software and its engineering → Abstraction, modeling and modularity**; *Software reverse engineering*; **Software evolution**; *Maintaining software*.

Keywords

software visualization, system comprehension, class archetypes

ACM Reference Format:
Mattia Giannaccari, *Supervised by Prof. Dr. Michele Lanza.* 2026. System Comprehension and Source Code Manipulation in eXtended Reality. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3774748.3787665



ICSE-Companion '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2296-7/2026/04
https://doi.org/10.1145/3774748.3787665

1 Introduction

Program comprehension is a fundamental activity in software engineering, consuming a large portion of developers' time and effort [18, 20, 27, 29]. Understanding complex software systems requires navigating multiple layers of abstraction, from architectural overviews to source code [3, 5, 21]. Over the past decades, research in software visualization has supported program comprehension by transforming abstract software artifacts into perceivable visual forms [6, 7, 19]. A number of studies have explored software visualization for guiding refactoring choices [2, 11], visualizing documentation [23], and supporting a number of software comprehension tasks, such as software metrics [16, 17, 25], the evolution of a system [4, 13, 24], software architecture [12, 14, 26, 31], small-scale software artifacts [8, 15, 17].

In parallel, advances in visualization technologies and computer graphics have expanded the ways in which software can be visually represented and explored. 3D visualizations offer spatial representations of software artifacts, making structural relationships and dependencies more intuitive to explore [30, 31]. Virtual and Augmented Reality (VR/AR) further enhance this experience by immersing developers in interactive environments to leverage spatial cognition, depth perception, and embodied interaction [14, 15, 22].

Interaction with source code has evolved from text-based editing to more intuitive and exploratory paradigms [2, 9]. Early work on linking visualization and editing demonstrated the benefits of bridging static and dynamic aspects of software behavior [8]. In immersive contexts, direct manipulation of visual elements can enable developers to refactor and navigate code through natural spatial interactions, reducing context switches and cognitive friction [10, 14]. Designing such interactions requires careful alignment between visual metaphors, code semantics, and user cognition.

Software visualization and interaction frameworks rely on data provided by Mining software repositories (MSR) practices. By extracting metrics, evolutionary, structural, and behavioral information from source code, MSR techniques enable accurate system representations [28]. The adoption of modular reverse-engineering platforms (*e.g.,* Moose [1]) and semantic analysis tools (*e.g.,* Language Servers,[1] CodeQL[2]) facilitates the construction of language-agnostic, queryable models of software systems. These models serve as the foundation for dynamic and interactive visual environments that evolve alongside the underlying codebase.

Building on these research areas, we envision a unified visualization and interaction framework for system comprehension and source code manipulation.

[1] **https://microsoft.github.io/language-server-protocol/**
[2] **https://codeql.github.com/**

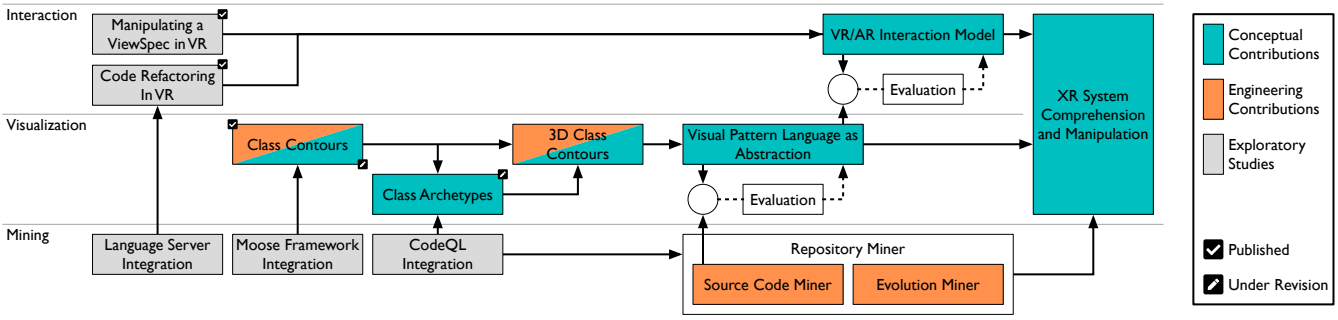


Figure 1: Research plan.

2 Research Focus

The overarching goal of my PhD research is to develop a conceptual framework for immersive software engineering that supports system comprehension and manipulation across diverse contexts. Our research addresses three interconnected aspects. Figure 1 illustrates the exploratory studies conducted, and the engineering and conceptual contributions we plan to investigate in pursuit of the ultimate goal. For each contribution, we also indicate which ones have already been published or submitted.

Mining. Source code and evolution mining is a necessary foundation for collecting the data to visualize and for developing a deep understanding of the domain. We are developing a flexible mining pipeline to support multiple programming languages, enabling the envisioned language-agnostic visualization framework.

Interaction and immersion. This aspect investigates how interactive environments, particularly virtual and augmented reality, can enhance system comprehension, facilitate information retrieval, and enable direct manipulation of code entities through cyber-physical interactions.

Visualization. This aspect focuses on designing a visual pattern language that defines mappings between software artifacts and visual metaphors (*e.g.,* buildings, districts). These metaphors encode software metrics as well as structural and behavioral properties.

Evaluation. For the main conceptual contribution, we plan to evaluate the visual pattern language with a focus on system comprehension, assessing how well its abstraction help developers understand system behavior, structure, and evolution. We then evaluate the interaction model, examining how it supports system comprehension and source-code manipulation.

3 XR System Comprehension and Manipulation

Our initial effort focused on exploring the interaction capabilities of VR systems. We identified a set of elementary actions that can be performed on a virtual 3D object (*e.g.,* grab, move) and modeled interactions as sequences of these actions. We then mapped these interactions to the corresponding operations in a target domain. We presented an approach for customizing a visualization through direct manipulation of a VR-native interface (*e.g.,* snapping together 3D objects), thus avoiding unnecessary context switches [10]. Later, we extended our set of interactions to source code manipulation [11]. In this context, we introduced a novel approach to refactor code by directly manipulating their visualization in VR.

Regarding the visualization aspect, we developed a novel visualization metaphor for representing the classes of a Java codebase using the *Class Contours* metaphor [11]. Each class is depicted as a two-dimensional architectural structure whose visual attributes correspond to class metrics and properties (*e.g.,* number and kind of methods and attributes, number of lines of code). Figure 2 shows an example of Class Contours. In a recent contribution, we analyzed the visual patterns emerging from Class Contours, showing that different types of classes (*e.g.,* data classes, utility classes) exhibit distinct visual characteristics. This work demonstrated that developers can infer class roles at scale from their visual characteristics, even without reading the source code.

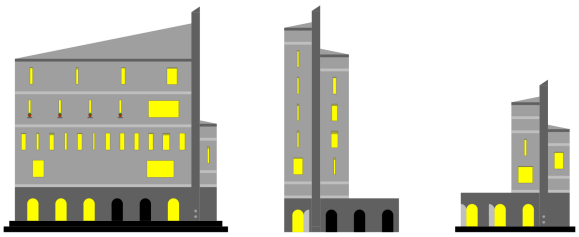


Figure 2: Three classes visualized with Class Contours.

We also started identifying visual patterns of Class Contours and mapping them to implementation patterns. By reifying these implementation patterns into detection rules, we define what we call *Class Archetypes*. We aim at encoding these archetypes into the visualization. First, we identify the class archetypes, then we use them as the basis for building their corresponding visual representations. This approach allows each class type to highlight relevant aspects of its implementation while omitting unnecessary details.

In parallel, we focused on mining codebases and developing a graph-based model to represent them. Our initial approach relied on parsing the source code using the Moose framework [1]. While Moose proved to be a powerful tool, it lacked support for capturing evolutionary information and for manipulating the source code. To address these limitations, we shifted to language servers, which allow us both to gather detailed codebase information and to trigger refactorings. In our most recent approach, we explored CodeQL, a semantic code analysis engine that treats source code as queryable data. By designing a custom set of queries, we are able to extract all the information necessary to populate and our graph model.

4 Research Agenda

The next steps of our research will follow four main directions.

Extending the metaphor. We plan to expand the catalog of class archetypes and design corresponding visual metaphors. We will define rules for mapping software metrics and structural properties to visual attributes that are both perceptually salient and semantically meaningful.

Enhancing interaction and immersion. We aim to integrate the visualization into a VR environment to support immersive exploration. Users will be able to interact with visual elements to retrieve information and trigger actions such as refactorings.

Cross-language visualization. We intend to develop an abstraction layer capable of parsing and representing code from multiple programming languages. Our goal is to ensure that the visual language remains consistent regardless of syntax or paradigm.

Evaluation. We plan to conduct a full evaluation to assess how effectively the visualization metaphor supports developers in system comprehension tasks. We will design a controlled experiment to measure the extent to which the visualization conveys architectural and evolutionary insights about a codebase. The study will investigate the impact of the metaphor on developers' ability to build accurate mental models of system structure and evolution. We will also examine whether VR/AR interaction and immersion enhance task performance by enabling developers to explore, analyze, and manipulate the system more quickly and accurately.

5 Conclusion

Our research aims to advance the field of software engineering by developing a unified visual and interactive language for program comprehension. By combining novel and expressive visual metaphors, immersive environments, and cross-language support, we envision a future in which developers can perceive, navigate, and manipulate software systems as if they were exploring a living city. Ultimately, our goal is to bridge the gap between source code and human cognition, enabling deeper understanding and more effective interaction with large and complex software systems.

References

[1] Nicolas Anquetil, Anne Etien, Mahugnon H. Houekpetodji, Benoit Verhaeghe, Stéphane Ducasse, Clotilde Toullec, Fatiha Djareddir, Jerôme Sudich, and Moustapha Derras. 2020. Modular Moose: A New Generation of Software Reverse Engineering Platform. In *Proceedings of ICSR 2020*. Springer, 119–134. https://doi.org/10.1007/978-3-030-64694-3_8

[2] Alex Bogart, Eman A. AlOmar, Mohamed W. Mkaouer, and Ali Ouni. 2020. Increasing the Trust In Refactoring Through Visualization. In *Proceedings of ICSEW 2020*. ACM, 334–341. https://doi.org/10.1145/3387940.3392190

[3] Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. 2004. *Object-Oriented Analysis and Design with Applications* (3rd ed.). Addison–Wesley.

[4] Michael Burch, Stephan Diehl, and Peter Weißgerber. 2005. EPOSee – A Tool For Visualizing Software Evolution. In *Proceedings of VISSOFT 2005*. IEEE, 1–2. https://doi.org/10.1109/VISSOF.2005.1684322

[5] Elliot J. Chikofsky and James H. Cross. 1990. Reverse Engineering and Design Recovery: A taxonomy. *IEEE Software* 7 (1990), 13–17. Issue 1. https://doi.org/10.1109/52.43044

[6] Noptanit Chotisarn, Leonel Merino, Xu Zheng, Supaporn Lonapalawong, Tianye Zhang, Mingliang Xu, and Wei Chen. 2020. A Systematic Literature Review of Modern Software Visualization. *J. of Visualization* 23 (2020), 539–558. Issue 4. https://doi.org/10.1007/s12650-020-00647-w

[7] Stephan Diehl. 2007. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software* (1st ed.). Springer.

[8] Stéphane Ducasse and Michele Lanza. 2005. The Class Blueprint: Visually Supporting the Understanding of Classes. *TSE* 31, 1 (2005), 75–90. https://doi.org/10.1109/TSE.2005.14

[9] Sarah Fakhoury, Devjeet Roy, Harry Pines, Tyler Cleveland, Cole S. Peterson, Venera Arnaoudova, Bonita Sharif, and Jonathan I. Maletic. 2021. gazel: Supporting Source Code Edits in Eye-Tracking Studies. In *Proceedings of ICSE 2021*. IEEE, 69–72. https://doi.org/10.1109/ICSE-Companion52605.2021.00038

[10] Mattia Giannaccari, Marco Raglianti, and Michele Lanza. 2024. Manipulating VR-Native User Interfaces for Software Visualization Customization. In *Proceedings of VISSOFT 2024*. IEEE, 111–115. https://doi.org/10.1109/VISSOFT64034.2024.00022

[11] Mattia Giannaccari, Marco Raglianti, and Michele Lanza. 2025. Code Refactoring in Virtual Reality. In *Proceedings of IDE 2025*. IEEE, 7–12. https://doi.org/10.1109/IDE66625.2025.00006

[12] Nathan Hawes, Stuart Marshall, and Craig Anslow. 2015. CodeSurveyor: Mapping Large-Scale Software to Aid in Code Comprehension. In *Proceedings of VISSOFT 2015*. IEEE, 96–105. https://doi.org/10.1109/VISSOFT.2015.7332419

[13] Elke F. Heidmann, Lynn von Kurnatowski, Annika Meinecke, and Andreas Schreiber. 2020. Visualization of Evolution of Component-Based Software Architectures in Virtual Reality. In *Proceedings of VISSOFT 2020*. IEEE, 12–21. https://doi.org/10.1109/VISSOFT51673.2020.00006

[14] Adrian Hoff, Lea Gerling, and Christoph Seidl. 2022. Utilizing Software Architecture Recovery to Explore Large-Scale Software Systems in Virtual Reality. In *Proceedings of VISSOFT 2022*. IEEE, 119–130. https://doi.org/10.1109/VISSOFT55257.2022.00020

[15] Adrian Hoff, Christoph Seidl, and Michele Lanza. 2023. Uniquifying Architecture Visualization through Variable 3D Model Generation. In *Proceedings of VaMoS 2023*. ACM, 77–81. https://doi.org/10.1145/3571788.3571798

[16] Danny Holten, Roel Vliegen, and Jarke J. van Wijk. 2005. Visual Realism for the Visualization of Software Metrics. In *Proceedings of VISSOFT 2005*. IEEE, 1–6. https://doi.org/10.1109/VISSOF.2005.1684299

[17] Michele Lanza and Stéphane Ducasse. 2001. A Categorization of Classes Based on the Visualization of Their Internal Structure: The Class Blueprint. In *Proceedings of OOPSLA 2001*. ACM, 300–311. https://doi.org/10.1145/504282.504304

[18] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of ICSE 2006*. ACM, 492–501. https://doi.org/10.1145/1134285.1134355

[19] Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. 2018. A Systematic Literature Review of Software Visualization Evaluation. *J. of Systems and Software* 144 (2018), 165–180. https://doi.org/10.1016/j.jss.2018.06.027

[20] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer–An Investigation of How Developers Spend Their Time. In *Proceedings of ICPC 2015*. IEEE, 25–35. https://doi.org/10.1109/ICPC.2015.12

[21] Gail C. Murphy, David Notkin, and Kevin Sullivan. 1995. Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. In *Proceedings of FSE 1995*. ACM, 18–28. https://doi.org/10.1145/222124.222136

[22] Gianlorenzo Occhipinti, Csaba Nagy, Roberto Minelli, and Michele Lanza. 2023. SYN: Ultra-Scale Software Evolution Comprehension. In *Proceedings of ICPC 2023*. IEEE, 69–73. https://doi.org/10.1109/ICPC58990.2023.00020

[23] Marco Raglianti, Csaba Nagy, Roberto Minelli, and Michele Lanza. 2022. DiscOrDance: Visualizing Software Developers Communities on Discord. In *Proceedings of ICSME 2022*. IEEE, 474–478. https://doi.org/10.1109/ICSME55016.2022.00062

[24] Mona Rahimi and Michael Vierhauser. 2022. Visualization of Aggregated Information to Support Class-Level Software Evolution. *J. of Systems and Software* 192, C (2022), 14 pages. https://doi.org/10.1016/j.jss.2022.111421

[25] Michele Risi and Giuseppe Scanniello. 2012. MetricAttitude: A Visualization Tool for the Reverse Engineering of Object Oriented Software. In *Proceedings of AVI 2012*. ACM, 449–456. https://doi.org/10.1145/2254556.2254643

[26] Satrio Adi Rukmono, Michel R. V. Chaudron, and Christopher Jeffrey. 2024. Layered BubbleTea Software Architecture Visualisation. In *Proceedings of VISSOFT 2024*. IEEE, 122–126. https://doi.org/10.1109/VISSOFT64034.2024.00024

[27] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. 2006. Questions Programmers Ask During Software Evolution Tasks. In *Proceedings of FSE 2006*. ACM, 23–34. https://doi.org/10.1145/1181775.1181779

[28] Mohamed Soliman, Michel Albonico, Ivano Malavolta, and Andreas Wortmann. 2025. Mining Software Repositories for Software Architecture – A Systematic Mapping Study. *Information and Software Technology* 181 (2025), 107677. https://doi.org/j.infsof.2025.107677

[29] Margaret-Anne Storey. 2005. Theories, Methods and Tools in Program Comprehension: Past, Present and Future. In *Proceedings of IWPC 2005*. IEEE, 181–191. https://doi.org/10.1109/WPC.2005.38

[30] Alfredo R. Teyseyre and Marcelo R. Campo. 2009. An Overview of 3D Software Visualization. *Transactions on Visualization and Computer Graphics* 15, 1 (2009), 87–05. https://doi.org/10.1109/TVCG.2008.86

[31] Richard Wettel and Michele Lanza. 2007. Visualizing Software Systems as Cities. In *Proceedings of VISSOFT 2007*. IEEE, 92–99. https://doi.org/10.1109/VISSOF.2007.4290706