

Collaborative Software Exploration with Multimedia Note Taking in Virtual Reality

Adrian Hoff

adho@itu.dk

IT University of Copenhagen, Denmark

Christoph Seidl

chse@itu.dk

IT University of Copenhagen, Denmark

Mircea Lungu

mlun@itu.dk

IT University of Copenhagen, Denmark

Michele Lanza

michele.lanza@usi.ch

Software Institute @ USI Lugano, Switzerland

ABSTRACT

Exploring and comprehending a software system, e.g., as preparation for its re-engineering, is a relevant, yet challenging endeavour often conducted by teams of engineers. Collaborative exploration tools aim to ease the process, e.g., via interactive visualizations in virtual reality (VR). However, these neglect to provide engineers with capabilities for persisting their thoughts and findings.

We present an interactive VR visualization method that enables (distributed) teams of engineers to collaboratively (1) explore a subject system, while (2) persisting insights via free-hand diagrams, audio recordings, and in-visualization VR screenshots.

We invited pairs of software engineering practitioners to use our method to collaboratively explore a software system. We observed how they used our method and collected their feedback and impressions before replaying their findings to the original developers of the subject system for assessment.

Video Demonstration—youtube.com/watch?v=32EIpf4V3b4

CCS CONCEPTS

• **Software and its engineering** → **Software maintenance tools; Maintaining software; Software reverse engineering.**

KEYWORDS

Software Visualization, Software Comprehension, Collaborative Software Engineering, Virtual Reality

ACM Reference Format:

Adrian Hoff, Mircea Lungu, Christoph Seidl, and Michele Lanza. 2024. Collaborative Software Exploration with Multimedia Note Taking in Virtual Reality. In *32nd IEEE/ACM International Conference on Program Comprehension (ICPC '24)*, April 15–16, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3643916.3644427>

1 INTRODUCTION

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICPC '24, April 15–16, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0586-1/24/04

<https://doi.org/10.1145/3643916.3644427>



Figure 1: Screenshot taken from a user's point of view while exploring a system. A collaborator takes notes on a virtual whiteboard.

Understanding a software system is a crucial task, frequently undertaken by groups of engineers [51]. For instance, it plays a vital role when preparing for the re-engineering of legacy software systems [45, 47], or for integrating a new member into a team. Nevertheless, the sheer size and complexity of a subject system or the engineers' proficiency with a programming language can hinder the software comprehension process. When attempting to explore and comprehend a software system by solely studying its source code, gaining an overview of its structure can be an intricate task.

Software visualization tools are available to assist engineers in gaining a comprehensive overview of a software system. These tools visually portray various facets of a system's structure, behavior, or evolution, offering an abstraction from the actual source code [10, 12, 20]. However, there are limited options for software visualizations that facilitate collaborative exploration. This becomes increasingly relevant in distributed developer teams [14, 28].

VR software visualization is an appropriate domain for collaboration, which is why state-of-the-art tools support it [29, 30]. However, existing approaches are limited, because they do not allow developers to take notes, posing a risk of valuable insights being lost.

We present a method that combines software exploration and note-making within a VR setting (see Figure 1). Engineers are immersed in an interactive 3D visualization of the source code of a subject system, facilitating collaborative engagement and comprehension of its structure. To document observations and insights in real-time, engineers have the capability to create virtual multimedia whiteboards (inspired by the work of Hoff et al. [24]). These

serve as a medium for pinning software elements from the visualization, drawing freehand diagrams with tool assistance, jotting down notes, as well as pinning recorded audio notes and captured in-visualization screenshots of the VR surroundings.

Additionally, these virtual whiteboards remain accessible and interactive outside the VR environment through synchronization with an Integrated Development Environment (IDE), enabling further examination and interaction, e.g., direct access to source code.

We used our method in an exploratory case study to investigate and report on how practitioners engage with VR tools for collaborative software exploration and, based on that, provide lessons learned for developers of similar tools. That is, we explored how engineers engage with exploration methods akin to ours.

RQ₁: *How do engineers explore and take notes of a software system in a collaborative VR software exploration and note taking tool?*

RQ₂: *What strengths and weaknesses do engineers perceive in using a collaborative VR software exploration and note taking tool?*

Further, we studied the results of engineers' explorations to provide first impressions on the suitability of the approach.

RQ₃: *What type of insights do engineers extract from a system when using a collaborative VR software exploration and note taking tool?*

To answer these questions, we invited two pairs of developers to use our method for exploring and comprehending a software system they had not seen before. We observed their exploration and note taking strategies (RQ₁), gathered their comments and opinions through questionnaires (RQ₂), and scrutinized the insights they accumulated on virtual whiteboards (RQ₃) by consulting with the original developers of the subject system, in order to verify the accuracy and relevance of the accumulated information.

Our results show that participants engaged vividly in a collaborative software exploration in VR, appreciating especially the architecture-level overview on the system's structure, yielding correct insights.

2 BACKGROUND AND RELATED WORK

Our method integrates collaborative software exploration and note-taking in a VR software visualization, positioning it within two main research domains: software visualization and (re-)documentation.

2.1 Software Visualization

Software visualization entails the use of visual metaphors to depict abstract and intangible concepts present in source code, with the intention of aiding users in obtaining both a general understanding and detailed insights into a software system's structural, behavioral, or evolutionary facets [10, 12, 20].

The visual metaphors employed range from abstract representations, such as graphs [6, 21, 32] or tree maps [25, 26], to real-world inspired structures like solar systems [19, 22, 41], islands [37], and cities [27, 35, 41, 48, 49]. Additionally, visual metaphors in software visualization can be categorized based on their dimensionality into 2D [1, 33, 36] and 3D variants, with the latter further distinguishable by the medium utilized, spanning standard 2D screens [46, 50, 52], VR [15, 23, 38], and AR [17, 34, 43].

There is a noticeable gap in research focusing on collaborative software exploration and understanding. Anslow et al. introduced

a method using an interactive touch-screen table for collaborative exploration, providing different structural views of a subject system [3]. This method is inherently designed for co-located collaboration. For distributed teams, research has explored VR as a medium for collaborative software exploration. Koschke et al. provide a collaborative VR method with diverse views, i.e., for clone detection or identifying architectural drift [28, 29]. Krause-Glau et al. propose a method for behavioral aspects investigation in a collaborative setting across various devices, including VR headsets [31]. While these contributions are valuable for collaborative VR software exploration, they lack the means for engineers to take notes on their insights, potentially leading to loss of valuable information during prolonged sessions – which constitute a relevant use case. We address this gap by proposing a method that integrates collaborative note taking into the exploration process (Section 3).

Collaborative environments for software exploration exist mainly for VR. They inadequately support concurrent note-taking, which is essential for preserving insights.

2.2 Software Documentation and Note Making

Various techniques and tools exist to aid engineers in generating notes and documentation on a software system's source code.

These range from automated documentation generators like RGB [39], Scribble [16], PAS [42], Re-Doc [2], and others [18, 40], to more informal methods like freehand sketching on paper or whiteboards [4, 11]. The latter is particularly prevalent in collaborative scenarios, providing a flexible medium for capturing ideas and insights. In recent work, Hoff et al. proposed a VR-based sketching method that enables engineers to pin elements from a software visualization and, based on that, sketch diagrams directly on whiteboards [24]. Their method performs conformance checks between the sketched diagrams and the source code, ensuring alignment. However, so far, the method proposed by Hoff et al. was used in only one study and that was not in a collaborative exploration setting. Further, to thoroughly capture complex and extensive notes during the software exploration process, we advocate for additional mechanisms beyond freehand sketching, which are not present in the previous work by Hoff et al.

While freehand sketching is a valuable tool for note-making in software engineering, current VR-based methods need more versatility to adequately capture long and complex notes.

3 COLLABORATIVE SOFTWARE EXPLORATION AND NOTE TAKING IN VR

To support teams of engineers in collaboratively exploring software systems, we propose a VR method that enables multiple engineers to enter a shared synchronized virtual environment (each with their individual head mounted VR device) to collaboratively explore a subject system while capturing thoughts and insights on shared VR whiteboards via freehand sketching, audio recordings, and in-visualization screenshots. Our method can be used in a local network as well as over the internet, allowing distributed teams of engineers to meet up in a virtual space to collaboratively explore a system's architecture and design, discuss ideas, and make notes.

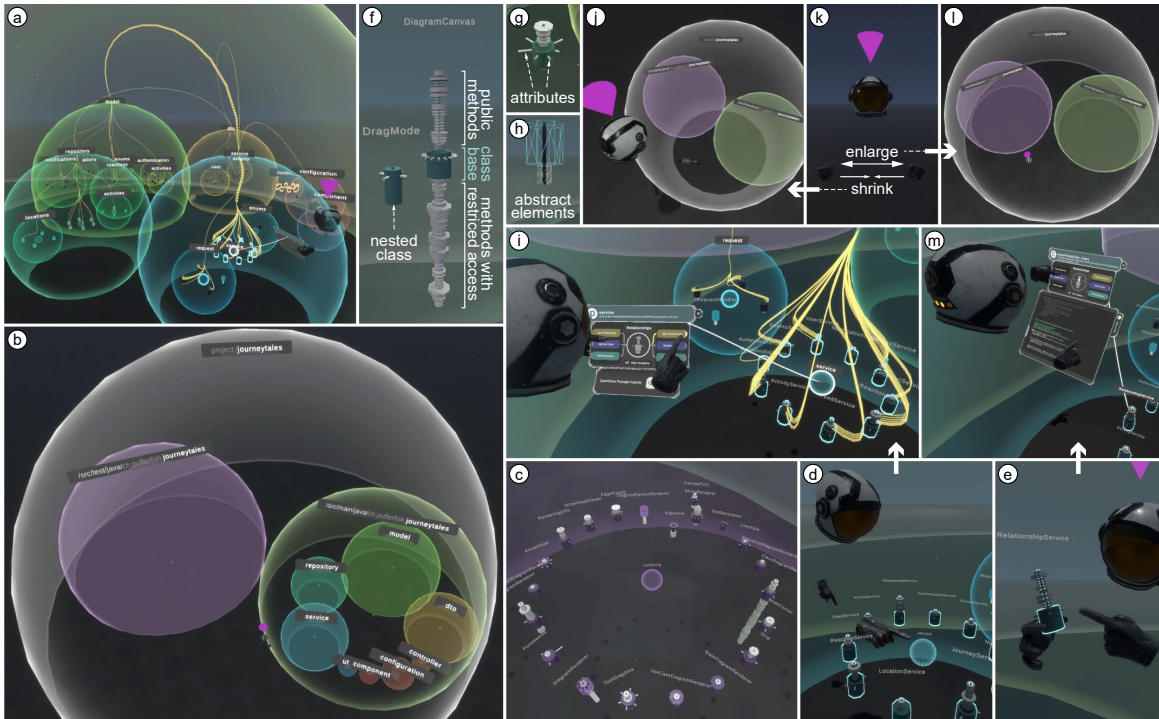


Figure 2: Implementation of our VR software visualization method. Folders/packages are represented as colored nested spheres (a-c). Classes, interfaces, etc. and their members are represented as stacks of cylinders (class cylinders), encoding meta information and structural metrics via location, shading, and form (f-h). Engineers can interact with elements (d, e) to blend in visual relationships graphs (i, a), read code (m), or scale the visualization up and down (j-l).

3.1 Collaborative Interactive VR Visualization

Our method supports teams of collaborating engineers in freely exploring software systems in a top-down fashion [13], following Schneiderman’s mantra “overview first, zoom and filter, details on demand” [44]. It provides engineers with an overview of the folder-level and class-level structure of a subject system, relationships between elements on both these levels, synchronized mechanisms for navigation and zoom, and details on demand on source code as text. Our concepts are designed for subject systems written in object-oriented programming languages. In the following, we elaborate on these using Java as an example.

3.1.1 Architecture Level - Folder Spheres. To provide engineers with an overview of the subject system, our method visualizes its folder structure as hierarchy of nested semi-transparent spheres (see Figure 2 (a)), similar to the software landscapes by Balzer et al. [7, 8]. Each sphere represents one folder with constituent elements (i.e., class-level elements and sub-folders) contained inside it, arranged in a circular layout parallel to the floor of the virtual environment ©. On system root level, all visual elements are contained in a root folder sphere containing the entire subject system (b).

Colors for Orientation. To help engineers with distinguishing elements from different parts of the subject system, our method determines colors for folder spheres based on the position of the represented folder in the system’s hierarchy.

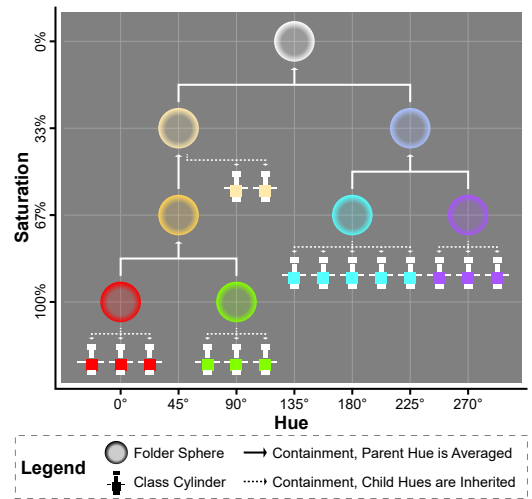


Figure 3: Example folder sphere hierarchy depicting the color scheme employed by our method.

Figure 3 depicts an example that illustrates the color distribution concept. Leaf-level folders are assigned evenly distributed hue values while their nesting level determines the color’s saturation. Hue values for all remaining folders result from the average of their sub-folder colors’ hue.

Interaction via Core. Depending on the amount and extents of their contained elements, folder spheres vary in size. With their semi-transparent look, they serve to guide engineers visually, creating an environment engineers can move through and stand in without unintentionally triggering interactions. We consistently manage interaction with folder spheres through a central core, see Figure 2 ©. These cores are uniform in size (independent of the folder sphere’s radius), ensuring a standard interaction across the system. Engineers have the option to grab cores for discussions with peers or tap on them to access a user interface offering additional information and options ④, such as selecting specific entity relationships to display. When an engineer releases the core, it automatically repositions itself back to the folder sphere’s center.

Opening/Closing Folder Spheres. To support engineers with focusing their investigation on relevant parts of a subject system, the visualization allows adjusting the amount of details in a view by opening and closing folder spheres. This is done via the folder sphere’s core in a user interface or via hand gestures. When opening a system for the first time, its root folder is opened by default with all first-level folders visible but closed (cf. Figure 2 ① and ②). Thereby, our method limits the amount of information immediately presented to engineers - a feature that is useful for large subject systems. Engineers open the visible folder spheres to look into their directly contained elements. Closing a sphere hides all constituent elements until opened again. The right-hand side of Figure 2 ③ shows a folder sphere after opening with its sub-folders spheres closed (compare with ① and ②). Opening and closing folder spheres is synchronized between all collaborating engineers.

While we considered a feature for opening all folders simultaneously, we decided against it to prevent users from becoming overwhelmed and losing their overview of the system. Our intention is to promote a top-down approach to exploration. Additionally, while our current implementation – favoring a simpler user interaction – does not allow engineers to completely hide folders and thereby only display selected non-hidden content, a feature like this could become valuable for navigating very large systems.

3.1.2 Class Level - Class Cylinders. Our method represents class-level elements in a way that provides engineers with a rapid overview of their inner member-level structure by encoding information in several visual properties: base cylinder, method cylinders, attribute spikes, and nested class cylinders discussed below.

Base Cylinder. Every class cylinder consists of one base cylinder that is colored according to the color of its containing folder (see Figure 3 on the coloring concept and Figure 2 ⑥ as implemented). Further, our method visually distinguishes between concrete and abstract class-level elements via the surface shading of the base cylinder. For instance, in Java, the base cylinder of a concrete class is displayed with an opaque surface ⑥ whereas abstract classes and interfaces receive a see-through wireframe surface to convey the look of a less tangible and less concrete structure ⑧.

Method Cylinders. Methods/functions of a class-level element are represented as cylinders where structural metrics determine their shape, i.e., the number of expressions in a method determines the cylinder’s height while cognitive complexity [9] (driven primarily by the depth of control flow splits) determines its radius.

To rapidly grant engineers an overview of the accessibility of methods, cylinders for methods with unrestricted access (e.g., public modifier in Java) are stacked on top of the base cylinder while cylinders for encapsulated methods (e.g., private, protected, or package visibility in Java) are stacked underneath the base cylinder. Figure 2 ⑦ depicts an example of a class with numerous public and private methods with varying size and complexity.

Attribute Spikes. Attributes/fields of a class-level element are represented as evenly distributed spikes originating from the colored base of a class cylinder, with unrestricted attributes being notably longer than encapsulated attributes, see Figure 2 ⑨.

Nested Class Cylinders. Nested class-level elements are represented using the above described mechanisms for regular classes, with two deviations: (1) their base cylinder is notably smaller, i.e., half the size of a root-level class; and (2) to represent the structural connection with their containing class-level element, nested class cylinders are arranged in an evenly spaced circle around the nesting class cylinder. As an example, the left-hand side of Fig. 2 ⑥ depicts a nested class “DragMode” in a regular class “DiagramCanvas”.

Interaction. Engineers in VR can grab class cylinders and hold them in their hands ③, e.g., to show them to a collaborator in a conversation. When released, the class cylinder smoothly returns to its original position in the visualization. Further, engineers can tip on the base cylinder, method cylinder, or attribute spikes of a class cylinder with their virtual fingers to open detailed views with additional information (see Sections 3.1.3 and 3.1.4).

3.1.3 Source Code Views. On demand, our method provides engineers with a textual view of a visualized element via a synchronized scrollable user interface, see Fig. 2 ④. This interface shows the source code of an element in the context of the file it is stored in. For instance, the source code view for a method displays all code of the containing file where it (1) initially scrolls to the location of the selected method and (2) emphasizes it by graying out all leading and trailing code. In Fig. 2 ④ the upper part of the presented code is grayed out to highlight the currently displayed code section.

3.1.4 Relationships. Our method provides engineers with an overview of the relationships between elements via an interactive graph based on statically analyzed references in the subject system’s code. This relationship graph represents references as animated directed lines between respective visual elements, originating from the member that contains the reference and ending in the referenced element, see Figure 2 ⑤ and ⑥. Our method distinguishes between incoming and outgoing references to or from a selected element as well as between type references, method calls, and field accesses respectively. Engineers can individually show and hide relationship lines for each of these categories for a selected folder, class-level element, method, or attribute via a synchronized user interface attached to the respective visual element, see Figure 2 ④. To reduce clutter and visual complexity, relationship lines are bundled together as they cross folder sphere boundaries, similar to the technique proposed in [7, 8], see Figure 2 ⑤. Displaying relationships for a software element containing multiple members (i.e., folders and class-level elements) summarizes all contained incoming/outgoing relationships.

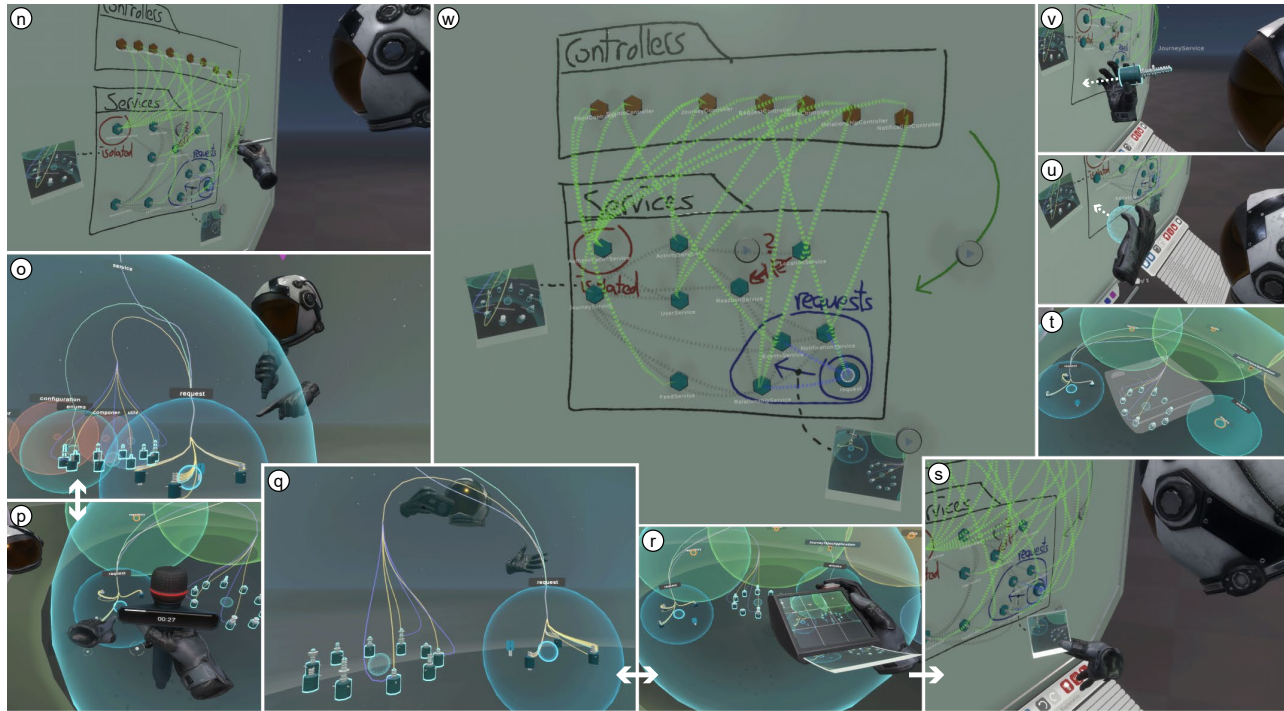


Figure 4: Implementation of our note making concepts in VR. While exploring a subject system, engineers can pick up a pen and freely sketch on virtual whiteboards (n). They can pin software elements from the visualization (v, u) and draw tool supported diagrams (w). To capture longer thoughts, they can create and pin sound recordings (o, p). Lastly, they can create and pin screenshots (q, r, s).

3.1.5 Navigation and Zoom. Engineers can freely navigate the visualization and change their point of view, which is synchronized in real time with all collaborators.

They can change their position by (1) teleporting through the virtual space and (2) rotating around their virtual axis (a VR mechanism commonly referred to as “snap turning” or “snap rotation”). To facilitate engineers’ potential experience with other VR tools or games, these resemble standard VR navigation mechanisms present in a majority of current VR applications. Further, they can use hand gestures to (a) move/offset the entire visualization (i.e., the hierarchy of folder spheres with all constituent elements) and (b) zoom in and out to change scale of the root folder sphere relative to the engineer’s hand position, see Figure 2 (j) to (l). To ensure a consistent point of view on the system, these mechanisms are synchronized between all collaborators. They offer fine-grain control over the point of view on a subject system while mitigating entry barriers by being operable when seated and in a small space.

3.2 Collaborative Note-Taking on VR Multi-Media Whiteboards


Our method encompasses mechanisms for collaborative note taking that enable engineers to persist insights and synchronize their understanding of a subject system while exploring it in VR: building on work from Hoff et al. [24], engineers work with virtual whiteboards to (a) pin elements from a VR software visualization (such as the folder spheres or class cylinders presented in Section 3.1) and (b) pick up a virtual pen and sketch freely. Figure 4 (n), (v) and (u)


illustrate these interactions. Software elements pinned to a whiteboard are represented as pins displaying references between the pinned software elements via curved relation lines. Further, the method presented in [24] automatically interprets freehand drawn shapes as outlines around pins (called modules) and relation-arrows between these. Based on that, it provides engineers with conformance checks between their sketches and the represented software structures by coloring relation lines between pins. Our method extends Hoff et al.’s work [24] with support for taking multi-media notes during an ongoing collaborative exploration via: 1) synchronized whiteboard interaction (3.2.1), 2) in-visualization screenshots (“VR-shots”) (3.2.2), and 3) audio recordings (3.2.3).



3.2.1 Synchronized Whiteboard Interaction. We extend the VR whiteboards presented in [24] by collaborative capabilities. For one, engineers can grab and freely position whiteboards, which is synchronized to their collaborators in real time. For another, we extend the virtual whiteboards with support for concurrent edits, see Figure 4 (n). Freehand drawn notes and pins on a whiteboard (including those presented in the remainder of this section) are synchronized in real time so that the content of all whiteboards is consistent across all collaborators.

3.2.2 In-Visualization Screenshots (“VR-Shots”). Our method enables engineers to quickly capture and pin screenshots to a whiteboard in VR, serving as a visual reference alongside other notes.

Engineers can take a screenshot by grabbing a virtual camera in VR and pressing its trigger, see Figure 4 (q) and (r). They can then

pin the virtual print-out of the screenshot to a shared whiteboard . For instance, this can be used to capture a view on relationship graph lines between two class cylinders in the visualization.

By tapping on pinned photos on a whiteboard, engineers revert the visualization to its exact state at the screenshot’s capture, including position, scale, folder spheres’ open/close statuses, and relationship graph state, while simultaneously being teleported to the screenshot’s location. A semi-transparent indicator marks the camera’s position when the photo was taken, aiding in understanding the photo’s perspective, see Figure 4 . This implements a form of save point for engineers to return to at later points in time.

3.2.3 Audio Recordings. Engineers can share longer and more detailed thoughts using audio recordings while exploring a subject system in VR, helping to free up their mental space. They do this by picking up a virtual microphone, speaking their thoughts aloud, and then pinning the recorded audio to a whiteboard as an audio pin, see Figure 4  and . By tapping on audio pins, engineers can play the recording, with playback shared in real-time among all collaborators.

3.2.4 Synchronization with IDE. With the above mechanisms, our method aims to support engineers in exploring and comprehending a subject system while persisting thoughts and insights on virtual whiteboards. Further, it synchronizes these whiteboards with an IDE to make engineers’ accumulated thoughts and insights accessible outside the VR environment. Figure 5 depicts an integration into the Eclipse IDE; the whiteboard from Figure 4 is displayed in the top-left area where engineers can zoom and pan, enlarge screenshots for detail inspection, play audio recordings, and directly open the code of pinned software elements in the IDE code editor. This feature bridges the gap between VR exploration and further use of created notes for subsequent work outside of VR, e.g., to implement planned changes.

4 CASE STUDY WITH PRACTITIONERS

We evaluated our method in an observational case study. Two pairs of software engineering practitioners participated, using an implementation of our VR method to explore a subject system in an open and uninterrupted setting (Figure 6, left-hand side). We collected their feedback through a post-session questionnaire. Subsequently, we scrutinized the insights they accumulated on virtual whiteboards and presented it to the original developers for validation of accuracy and relevance (Figure 6, right-hand side).

4.1 Tool Implementation

We implemented the concepts presented in Section 3 in our tool Immersive Software Archaeology¹ (ISA). ISA employs a client-server architecture, with the server ensuring real-time synchronization and consistency of whiteboards and the visualization across clients. The client is developed in C# with the Unity game engine, relying on the SteamVR platform² to make it compatible with all major VR hardware. The server side is implemented in Java as part of ISA’s ecosystem of Eclipse plug-ins where it integrates with an automated software analysis. All code is open-source in ISA’s repository¹.

¹<https://gitlab.com/immersive-software-archaeology>
²<https://store.steampowered.com/app/250820/SteamVR/>

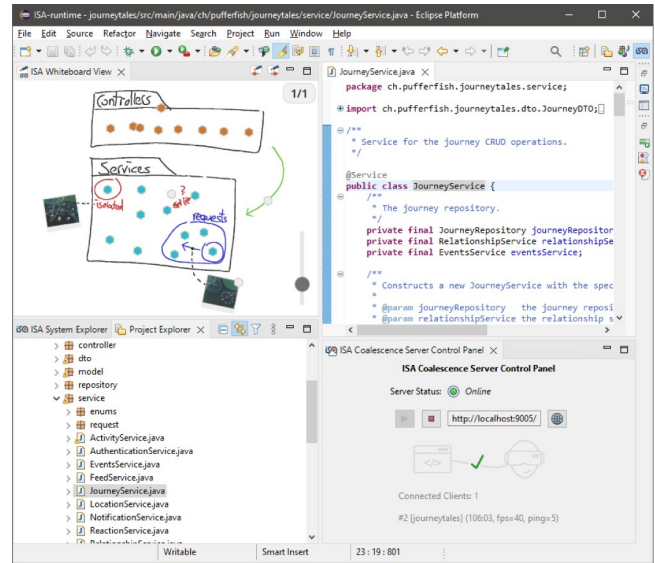


Figure 5: VR whiteboards are synchronized with an IDE where they can be inspected and interacted with (top-left area) by zooming, panning, playing audio recordings, and opening pinned elements.

4.2 Case Study Procedure

Our study is divided in two phases (cf. Figure 6), i.e., VR sessions with software engineering practitioners (depicted on the left) and subsequent result validation by the original developers of the subject system (shown on the right).

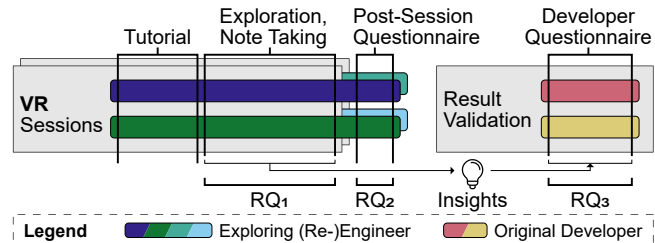


Figure 6: Case study procedure: Two pairs of developers collaboratively explore a subject system and fill in a questionnaire (left-hand side). We relay the gathered insights to the original developers and assess their correctness and relevance (right-hand side).

4.2.1 VR Sessions. In a first step, we addressed RQ₁ and RQ₂ through VR sessions with pairs of software engineering practitioners. Our tool supports both internet-based distributed setups as well as local network connections. However, for our study, we chose to have participants collaborate in the same room to facilitate a smoother introduction and especially to help with the VR hardware. The VR sessions were structured in three main stages.

1. *Tutorial (max. 30 minutes).* Each VR session began with a brief tutorial, explaining the tool’s visual metaphor and VR controls, navigating the system and interacting with elements. Participants and the experiment instructor (first author of this paper) went

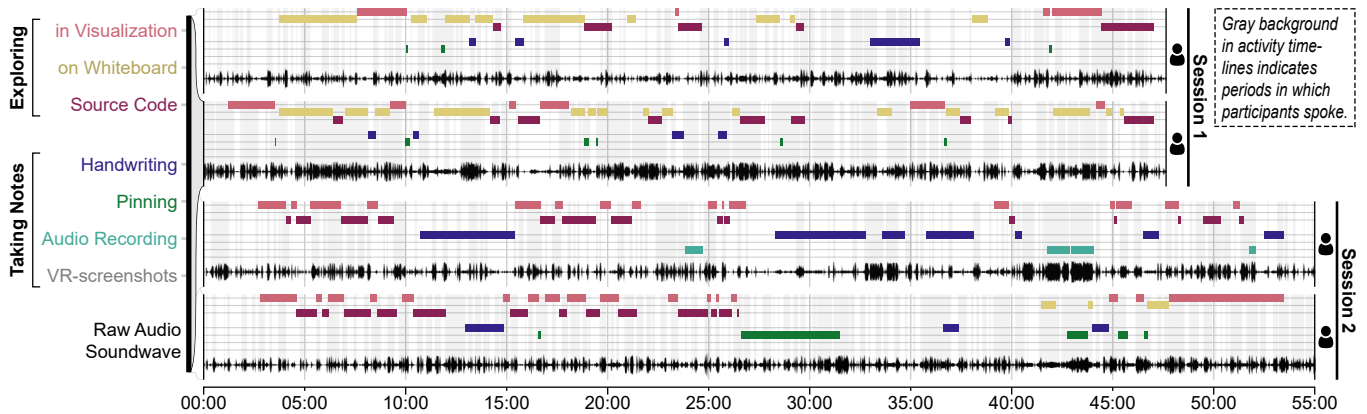


Figure 7: Timeline depicting participant activity in the two VR sessions. It shows for each participant the intervals in which they were carrying out one of the activities listed on the left-hand side. Note that activity labels are magnified for one of the four participants due to space constraints. Further, participants’ speech recordings are visualized, with intervals of actively speaking to one another highlighted.

through this process together in VR, following a predefined script. The full tutorial script can be found in our online appendix³).

2. *Exploration and Note Taking (min. 45 minutes).* Following the tutorial, the instructor briefly introduced the subject system (two sentences) and read out the open experiment task: “Please explore the system and make notes of your findings.” Subsequently, the instructor did not intervene unless there were technical issues or if participants requested help with controls or interaction. No content-related assistance was given. The entire script is available in our appendix³). The end result of each session was a set of VR whiteboards with notes and audio recordings (see online appendix³). Additionally, for both sessions, we video-recorded participants’ VR point of view as well as audio of what they were saying.

3. *Post-Session Questionnaire.* After the VR sessions, each participant filled out an anonymous questionnaire on their own, sharing their thoughts on the support provided by our method in collaboratively exploring a software system and taking notes on gained insights. The full questionnaire is available in our online appendix³.

4.2.2 *Result Validation.* In the second phase, we addressed RQ₃ by forwarding participants’ insights from the VR sessions to the subject system’s original developers using the post-session questionnaire³. For that purpose, we manually analyzed all handwritten notes and audio recordings on the virtual whiteboards created by participants of the previously conducted VR sessions, consulting video recordings of the VR sessions to ensure accurate context interpretation. We combined video recordings of participants’ VR POV side-by-side, resulting in one video for each session. We then manually analyzed these videos and extracted all notes written on virtual whiteboards using the context provided by the video, e.g., a conversation between participants. We included all insight explicitly noted and display them in Table 2. Long audio pins commenting on multiple aspects resulted in multiple separate insights. The subject system’s original developers subsequently evaluated each insight for its accuracy and relevance (“How relevant is this insight when planning potential changes to the system?”).

³<https://doi.org/10.6084/m9.figshare.24499726>

4.3 Subject System

Our study focuses on a Java Spring Boot web server backend API, a component of a travel journey management system with approximately 10,000 lines of code. The system enables users to monitor their travel activities, whether by plane, train, or visits to specific locations. Key features include persistent access to travel data, a user account system, search functionality, and friend management. The system’s source code is publicly accessible⁴.

We chose this system for our study because we could contact the original developers, a crucial aspect for validating our results. Moreover, Spring is among the most popular frameworks for building enterprise applications and thus the system is a good representative for a large class of relevant Java systems. However, note that our visualization is not tailored for Spring applications.

4.4 Participants

Our only inclusion criterion for participants of the study was having professional experience with Java development. We did not explicitly seek participants experienced with Spring Boot or VR and asked about contact with these technologies in our questionnaire. Four engineers from one company participated in our study, all with limited VR experience prior to the study. One participant reported having never used VR at all. Concerning their familiarity with software systems akin to the subject system, three of the participants confirmed their regular use of the Spring platform, while one reported having previous, yet not extensive, experience with it.

4.5 RQ₁: How do engineers explore and take notes of a software system in a collaborative VR software exploration and note taking tool?

Figure 7 depicts a timeline highlighting the activities and communication patterns of each participant in the study. We analyzed video recordings for both sessions, categorizing participant activities into seven distinct types: (1) exploring the system through inspecting elements of the visualization (most notably folder spheres, class

⁴gitlab.com/usi-si-oss/teaching/projects-showcase/sa4/team-4-pufferfish/backend

cylinders, relationship graphs), (2) exploring relationships between elements on a virtual whiteboard (mostly re-arranging pins and tracing relations via curved relation lines between pins), (3) examining and navigating source code on UI canvases, (4) handwriting notes on a virtual whiteboard, (5) attaching software elements to a whiteboard, e.g., in proximity to a handwritten note, (6) creating and pinning audio recordings to a virtual whiteboard, and (7) capturing and pinning VR screenshots to a virtual whiteboard.

We summarize multiple interactions with the visualization under point (1) since retrospectively determining the participants' focus during their visual exploration of a complex view comprised of nested, semi-transparent folder spheres, numerous class cylinders, and connecting relation lines is challenging to accurately pinpoint.

Varied Architectural Exploration. Throughout both sessions, participants oscillated between exploring the overarching architecture and inspecting source code details, mostly engaging in collaborative work with occasional individual exploration. In Session 1, participants adopted a top-down, breadth-first approach for exploring the system's architecture, employing a mix of examining the visual elements of the visualization (i.e., folder spheres and class cylinders) and pinning package folders to virtual whiteboards for continued examination. They inspected source code when class-level elements aroused their interest, though their primary focus remained on comprehending the system's architectural structure. This pattern is visible in the activity timeline for Session 1, as shown in Figure 7.

Participants in Session 2 adopted a more dynamic approach, swiftly navigating through folder spheres and intermittently analyzing source code, driven by the spontaneous discovery of relevant folders and classes. In Session 2, virtual whiteboards were scarcely used for exploration purposes (cf. Figure 7).

Handwriting Varied Across Sessions. In both sessions, participants made extensive use of handwriting for documenting insights, but varied their strategies. Session 1 participants mainly wrote isolated single words to provide context to clusters of pinned class-level elements on the whiteboard, while participants in Session 2 produced more elaborate handwritten notes and supplemented them with audio recordings. In both cases, whiteboard pins for software elements and audio recordings were strategically located in relation to handwritten notes.

VR Screenshots Not Utilized. None of the participants incorporated VR screenshots in their virtual whiteboards. As far as we can tell, participants did not experience scenarios in which they perceived capturing an image of a specific part of the visualization or using it as a save point for future reference as beneficial.

Ample Communication. Figure 7 shows for each participant both their raw audio soundwave on our recordings of the VR session as well as periods of time when they were speaking (as gray background behind the activity timelines). It shows that in both sessions, participants were overwhelmingly active in communication with only occasional short silent phases of individual source code inspection. Furthermore, we observed discussion phases, where exploration and note taking were temporarily halted to deliberate on ideas, most notably in Session 1 ca. from minute 30 to 33.

Table 1: Stacked bar chart with participant responses from the post VR session questionnaire. Bars are colored and sorted by participant (see mapping of participants to colors in Figure 6). Bars extending more to the right indicate stronger agreement or perceived usefulness. The answer of Participant 3 to Q₇ was discarded due to a misunderstanding of the question.

No.	Statement / Question	Agreement (Likert-Scale)	Avg.
Q ₁	The VR tool helped me with exploring the subject system's architecture.	5 4 4 4	4.25
Q ₂	The VR tool helped me with exploring the system's class-level elements.	3 5 2 3	3.25
Q ₃	The VR tool helped me with taking notes and documenting the subject system.	2 3 2 4	2.75
Q ₄	How valuable do you assess the audio recording feature?	2 2 4 5	3.25
Q ₅	How valuable do you assess the camera feature?	4 5 2 4	3.75
Q ₆	How valuable do you assess the whiteboard as a whole?	4 5 5 4	4.5
Q ₇	How valuable do you assess the collaborative aspect of the VR tool?	4 5 4	4.33
Q ₈	The VR tool provided a benefit to how I would usually have explored and taken notes on the software system.	2 4 4 5	3.75

4.6 RQ₂: What strengths and weaknesses do engineers perceive in using a collaborative VR software exploration and note taking tool?

In the following, we provide a summary of the feedback collected from participants of both VR sessions through a post-session questionnaire (cf. Figure 6). The questionnaire yielded participants' quantitative verdict on the support they received for exploring a subject system and taking notes on the results in VR (Table 1) as well as qualitative data in terms of free text comments. A full version with all verbatim comments is available in our online appendix³.

Exploration. Table 1 shows that participants assessed the support they received in exploring significantly higher than the support they received in taking notes (Q₁&Q₂ vs. Q₃).

Further, Table 1 shows that participants valued the exploration capabilities in VR especially for architecture-level aspects (Q₁ vs. Q₂). In their comments, participants reported on a perceived ease in obtaining an overview of the subject system, identifying relevant software elements, and understanding their interconnections. Apart from that, they wished for textual search for software elements and reported on a general unfamiliarity with VR resulting in perceived slow interaction with the tool: "I'm not used to VR, so I was slow to perform the activities."

Note Taking. Participants' overall merit of using VR to take notes was mixed (cf. Table 1; Q₃). On a positive side, they highlighted the benefits of having unlimited space on the virtual whiteboards, their integration with the rest of the visualization, and the resulting high-level views on a subject system's architecture. One participant particularly emphasized the voice recordings and the ease of linking audio to visual elements on the whiteboards. Conversely, criticisms centered around the cumbersome nature of handwriting in VR with

multiple related comments to an unfamiliarity with VR and the need for more practice.

For audio recording, participants of Session 1 who did not use the feature gave worse feedback (both giving a 2 on the Likert-scale, Q₄) than participants of Session 2 (who extensively used the audio recordings, and who gave a 5 and 4 grade respectively).

Session 1 participants both justified their low scores with the absence of an automated speech-to-text transcription feature, e.g., “I wouldn’t use it much because I prefer having written notes. Maybe it would be useful if I could create a transcript from the recording”.

In accordance with our observations during the VR sessions, participants perceived the virtual camera as underutilized, yet appreciated (cf. Table 1; Q₅): “I forgot to use it, but it certainly helps to explore/move faster when switching between whiteboards and the codebase”.

Overall, despite comments on requiring more practice to feel comfortable with handwriting in VR, participants assessed the virtual whiteboards as very useful (Q₆).

Collaboration. Feedback on VR collaboration was mostly positive (cf. Table 1; Q₇) and in line with our observations and answer to RQ₁. As suggestions for future work, participants emphasized a wish for locking elements in space that are shared between collaborators (especially whiteboards) so that they cannot be moved and repositioned until unlocked again to avoid accidental interactions.

4.7 RQ₃: What type of insights do engineers extract from a system when using a collaborative VR software exploration and note taking tool?

Table 2 lists all insights on the subject system captured by participants in form of handwritten notes and audio recordings on virtual whiteboards. Session 1 yielded Insights I_{1,1} to I_{1,4}, while Session 2 provided Insights I_{2,1} to I_{2,10}. We investigated these insights to discern patterns and verified their accuracy and relevance by seeking feedback from the original developers of the subject system via an online questionnaire (available in our online appendix³).

Extracted insights. Session 1 participants focused on the system’s overarching structure, adopting a strict top-down approach (cf. answer to RQ₁ above). This pattern is evident in their notes, which exclusively covered system-level and architectural aspects without addressing behavioral details.

Conversely, Session 2 participants adopted a use-case centered exploration, focusing on more specific aspects of the system’s behavior and inner workings rather than system-wide aspects. They started their exploration with the system’s test package, e.g., investigating example usages of different parts via unit tests. Thus, notes of Session 2 participants touched upon testing (Insights I_{2,2} and I_{2,3}) while the remaining notes capture the system’s behavior from a user’s point of view.

Correctness. As per verdict of the original developers, the insights participants noted during their VR sessions were largely correct with an average of 4.43 on a scale from 1 (incorrect) to 5 (correct). Only Insight I_{2,8} was clearly incorrect.

Relevance. As per the verdict of the original developers of the subject system, the relevance of participants’ insights varied with an average of 3.61 on a scale from 1 (irrelevant) to 5 (relevant).

Table 2: Stacked bar chart for feedback from the subject system’s original developers on correctness and relevance of the insights collected during both VR sessions. Each bar represents an answer from one of the two original developers. Wider bars with higher values indicate more correctness or perceived relevance.

No.	Participants’ insights into the subject system, noted on virtual whiteboards	Feedback from Developers			
		Correctness		Relevance	
I _{1,1}	Among the system’s modules are ‘controller’ and ‘service’.	5	3	5	2
I _{1,2}	The system uses server-side events for social media aspects, e.g., updating the trip feed, forwarding reactions to trips, etc.	5	5	3	3
I _{1,3}	All controllers interact with the authentication service.	4	4	4	5
I _{1,4}	The location service is isolated from the other services (they don’t interact).	4	5	3	2
I _{2,1}	The backend system does not have any sub-systems, it is a single big system with multiple responsibilities.	4	4	5	4
I _{2,2}	The system has both unit tests and integration tests.	4	5	5	5
I _{2,3}	There are todos in the test classes that should be looked into.	5	5	2	2
I _{2,4}	Journeys consist of lists of activities with visits to locations.	5	5	5	5
I _{2,5}	Users create and own journeys.	5	5	5	3
I _{2,6}	User profiles can be private or public.	5	5	5	4
I _{2,7}	Users can add one another as friends.	5	5	5	4
I _{2,8}	Friends can have trips together.	1	1	4	1
I _{2,9}	Users can react to trips of friends.	5	5	1	3
I _{2,10}	There is a feed of activities (trips) that is shared with friends.	5	5	4	2

4.8 Discussion of Results and Lessons Learned

In the following, we summarize our results for RQ₁-RQ₃ and highlight lessons learned from our case study for builders of related VR tools. Overall, our study demonstrates that even practitioners with minimal prior experience in VR software visualization can utilize methods like ours for collaborative exploration and comprehension.

Key Suitability at Architecture-Level. Through observation and direct participant feedback, we identified that a VR exploration and multimedia note-taking environment primarily enhances work at the architectural level as opposed to finer statement-level details.

Freehand Sketching in VR Requires Practice. Based on participant actions and feedback, relying solely on handwriting and sketching in VR for capturing insights can be tedious and time-consuming, a situation that might extend beyond VR environments. However, participants also noted improvements in their VR handwriting skills even within the short duration of the case study sessions, an observation made frequently in conversations between them, e.g., “I wrote ‘friends’. Ah, the handwriting is getting better!”.

Audio Recordings Require Transcription. Audio recordings were more favorably received than handwriting, although there was a

clear desire for enhanced tool support in this area, mostly in terms of automatic conversion of audio recordings to text.

VR Screenshots Require Further Investigation. While screenshots were deemed valuable, they were not utilized in the VR sessions. Additional research is necessary to explore this phenomenon, e.g., to investigate a correlation with the duration of VR sessions or in usage spanning over multiple VR sessions.

Communication was Ample. There were clear signs of successful collaboration among participants. This was evident not only from our observations during the VR sessions but also from the constant communication occurring throughout them, as depicted in the audio waves in Figure 7. Participants utilized having different perspectives on the same subject system and synchronizing their insights, e.g., one participant reading code, the other finding a mentioned method through the relationship graph and exploring from there.

Accurate Results with Varying Relevance. In addressing RQ₃, we determined that participants in our case study produced results that were accurate. The relevance of these results varied, which is understandable considering that we instructed participants to record all noteworthy findings without specifically evaluating their relevance in the constrained timeframe of the VR sessions. Although our findings on the correctness of participants' insights are encouraging, further studies are needed to investigate their relevance, e.g., to find correlations between insights, their correctness, and their relevance (which we were not able to find in this work).

VR Requires Practice. A recurring theme in feedback and observations was the novelty and necessary practice associated with both the method and VR as a technology in general. This applies to both exploring software in VR and taking notes on the findings as highlighted in participant feedback during the post-session questionnaires.

4.9 Reflections and Threats to Validity

In the following, we discuss potential risks to our study's findings and our strategies to reduce their impact.

Participant Selection and Number. The preliminary case study presented in this work was conducted with only four engineers organized in two pairs that worked together. These were selected by convenience, i.e., we contacted a company and asked for volunteering engineers willing to participate in the study. This indicates a necessity for more extensive investigations to uncover additional patterns in the behavior of a broader spectrum of engineers. Nevertheless, our study granted first insights into how practitioners use a VR software exploration and note taking tool, how they assess different tool features, what kind of information they extract from it, and how accurate and relevant these are.

Subject System. The subject system used in our study comprises 10,000 lines of code. Results from our study must thus be interpreted in a context of exploring similarly sized systems.

Further studies must be conducted to evaluate the scalability of VR exploration and note taking tools for systems of significantly larger scale. We opted for the subject system used in this study because we had access to its original developers, a unique opportunity for assessing the results of participants' exploration sessions.

Potential Response Biases. Response biases in studies can occur due to question wording or social dynamics among participants or between participants and interviewers [5]. In our study, this threat potentially applies to feedback from VR participants as well as to the result assessments of the subject system's original developers. To counteract response biases, we kept our experiment's purpose confidential until after its completion and minimized our interaction with both participants and the subject system's original developers. With the latter, we had no contact prior to the study other than for forwarding the insights collected by VR participants (initial contact was established by a third party). Moreover, we gathered feedback through anonymous, individually answered questionnaires. VR participants completed these directly following their session.

5 CONCLUSION AND FUTURE WORK

Our method allows distributed software engineering teams to immerse into an interactive VR space to visually analyze and comprehend a subject software system. In this virtual space, engineers can take multi-media notes on their observations and insights using a variety of tools including freehand sketching, diagramming, audio recordings, and screenshots.

We conducted a preliminary case study to investigate how pairs of software engineering practitioners use our method to explore a subject system. The participants, new to the system they were exploring, provided valuable insights and feedback on different VR features. Further, we assessed the accuracy and relevance of their findings by consulting the system's original developers. All in all, despite requiring more practice to fully utilize our method and VR technology, participants found our approach beneficial for architectural exploration, exhibited vivid communication during the sessions, and produced correct notes.

Future work will entail larger and more refined studies with additional software engineering practitioners, and comparisons with traditional (non-VR) collaborative software exploration methods such as other kinds of software visualization and IDEs. Further, we plan to study engineers' interaction with exported notes in the IDE (such as the one displayed in Figure 5). We also plan to enhance our method based on participant feedback, especially on integrating audio recordings with speech-to-text transcription and VR-friendly search capabilities (e.g., also via a speech interface).

As a bottom line, it is crucial to interpret our findings within the current technological landscape. Our participants were navigating a novel technology and tool set. As VR technology becomes more mainstream and engineers become more accustomed to similar tools, we anticipate that the results and user experiences reported here will improve.

To conclude, we remain optimistic about the potential of VR methods to enhance collaborative software engineering practices.

ACKNOWLEDGMENTS

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

REFERENCES

- [1] Mohammad Alnabhan, Awni Hammouri, Mustafa Hammad, Mohammed Atoum, and Omamah Al-Thnebat. 2018. *2D visualization for object-oriented software systems*. <https://doi.org/10.1109/ISACV.2018.8354085> Pages: 6.
- [2] Nicolas Anquetil, Káthia Oliveira, Anita Paulo, Laesse jr, and Susa Vieira. 2005. A tool to automate re-documentation. (2005).
- [3] Craig Anslow, Stuart Marshall, James Noble, and Robert Biddle. 2013. SourceVis: Collaborative software visualization for co-located environments. In *2013 First IEEE Working Conference on Software Visualization (VISOFT)*. IEEE, Eindhoven, Netherlands, 1–10. <https://doi.org/10.1109/VISOFT.2013.6650527>
- [4] Sebastian Baltes and Stephan Diehl. 2014. Sketches and Diagrams in Practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 530–541. <https://doi.org/10.1145/2635868.2635891> arXiv:1706.09172 [cs].
- [5] Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering* 27, 4 (2022), 94.
- [6] M. Balzer and O. Deussen. 2004. Hierarchy Based 3D Visualization of Large Software Structures. In *IEEE Visualization 2004*. IEEE Comput. Soc, Austin, TX, USA, 4p–4p. <https://doi.org/10.1109/VISUAL.2004.39>
- [7] M. Balzer and O. Deussen. 2007. Level-of-detail visualization of clustered graph layouts. In *2007 6th International Asia-Pacific Symposium on Visualization*. IEEE, Sydney, NSW, 133–140. <https://doi.org/10.1109/APVIS.2007.329288>
- [8] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. 2004. Software landscapes: Visualizing the structure of large software systems. In *IEEE TCVG*.
- [9] G Ann Campbell. 2018. Cognitive complexity: An overview and evaluation. In *Proceedings of the 2018 international conference on technical debt*. 57–58.
- [10] Pierre Caserta and O Zendra. 2011. Visualization of the Static Aspects of Software: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 17, 7 (July 2011), 913–933. <https://doi.org/10.1109/TVCG.2010.110>
- [11] Uri Dekel and James D Herbsleb. 2007. Notation and Representation in Collaborative Object-Oriented Design: An Observational Study. (2007).
- [12] Stephan Diehl. 2007. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media.
- [13] S. Ducasse and D. Pollet. 2009. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE Transactions on Software Engineering* 35, 4 (July 2009), 573–591. <https://doi.org/10.1109/TSE.2009.19>
- [14] Christof Ebert, Marco Kuhmann, and Rafael Prikladnicki. 2016. Global Software Engineering: Evolution and Trends. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*. IEEE, Orange County, CA, USA, 144–153. <https://doi.org/10.1109/ICGSE.2016.19>
- [15] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. 2015. Exploring software cities in virtual reality. In *2015 IEEE 3rd Working Conference on Software Visualization (VISOFT)*. IEEE, Bremen, Germany, 130–134. <https://doi.org/10.1109/VISOFT.2015.7332423>
- [16] Matthew Flatt, Eli Barzilay, and Robert Bruce Findler. 2009. Scribble: Closing the Book on Ad Hoc Documentation Tools. (2009).
- [17] Dussan Freire-Pozo, Kevin Cespedes-Arancibia, Leonel Merino, Alison Fernandez-Blanco, Andres Neyem, and Juan Pablo Sandoval Alcocer. 2023. DGT-A Visualizing Code Dependencies in AR. In *2023 Working Conference on Software Visualization (VISOFT)*. IEEE.
- [18] Verena Geist, Michael Moser, Josef Pichler, Stefanie Beyer, and Martin Pinzger. 2020. *Leveraging Machine Learning for Software Redocumentation*. <https://doi.org/10.1109/SANER48275.2020.9054838> Pages: 626.
- [19] Hamish Graham, Hong Yul Yang, and Rebecca Berrigan. 2004. A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics. (2004), 7.
- [20] Denis Gračanin, Krešimir Matković, and Mohamed Eltoweissy. 2005. Software visualization. *Innovations in Systems and Software Engineering* 1, 2 (Sept. 2005), 221–230. <https://doi.org/10.1007/s11334-005-0019-8>
- [21] O. Greevy, M. Lanza, and C. Wyseier. 2005. Visualizing Feature Interaction in 3-D. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, Budapest, Hungary, 1–6. <https://doi.org/10.1109/VISOFT.2005.1684317>
- [22] Adrian Hoff, Lea Gerling, and Christoph Seidl. 2022. Utilizing Software Architecture Recovery to Explore Large-Scale Software Systems in Virtual Reality. In *2022 Working Conference on Software Visualization (VISOFT)*. IEEE, Limassol, Cyprus, 119–130. <https://doi.org/10.1109/VISOFT55257.2022.00020>
- [23] Adrian Hoff, Michael Nieke, and Christoph Seidl. 2021. 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*. ACM, Athens Greece, 1455–1458. <https://doi.org/10.1145/3468264.3473128>
- [24] Adrian Hoff, Christoph Seidl, Mircea Lungu, and Michele Lanza. 2023. Preparing Software Re-Engineering via Freehand Sketches in Virtual Reality. In *Proceedings of the 39th IEEE International Conference on Software Maintenance and Evolution*. IEEE. <https://www.inf.usi.ch/lanza/Downloads/Hoff2023b.pdf>
- [25] Danny Holten, Roel Vliegen, and Jarke van Wijk. 2006. Visualization of Software Metrics using Computer Graphics Techniques. (Jan. 2006).
- [26] Danny Holten, Roel Vliegen, and Jarke J. Van Wijk. 2005. Visual Realism for the Visualization of Software Metrics. In *In Proceedings of Visualizing Software for Understanding and Analysis (VISOFT 2005)*. Springer, 1–6.
- [27] C. Knight and M. Munro. 2000. Virtual but visible software. In *2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics*. IEEE Comput. Soc, London, UK, 198–205. <https://doi.org/10.1109/IV.2000.859756>
- [28] Rainer Koschke and Marcel Steinbeck. 2021. Modeling, Visualizing, and Checking Software Architectures Collaboratively in Shared Virtual Worlds. (2021).
- [29] Rainer Koschke and Marcel Steinbeck. 2021. SEE Your Clones With Your Teammates. In *2021 IEEE 15th International Workshop on Software Clones (IWSC)*. IEEE, Luxembourg, 15–21. <https://doi.org/10.1109/IWSC53727.2021.00009>
- [30] Alexander Krause-Glau, Marcel Bader, and Wilhelm Hasselbring. 2022. *Collaborative Software Visualization for Program Comprehension*. <https://doi.org/10.1109/VISOFT55257.2022.00016> Pages: 86.
- [31] Alexander Krause-Glau, Malte Hansen, and Wilhelm Hasselbring. 2022. Collaborative program comprehension via software visualization in extended reality. *Information and Software Technology* 151 (Nov. 2022), 107007. <https://doi.org/10.1016/j.infsof.2022.107007>
- [32] M. Lanza and S. Ducasse. 2003. Polymetric views - A lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering* 29, 9 (Sept. 2003), 782–795. <https://doi.org/10.1109/TSE.2003.1232284>
- [33] Mircea Lungu, Michele Lanza, and Oscar Nierstrasz. 2014. Evolutionary and collaborative software architecture recovery with SoftwareNaut. *Science of Computer Programming* 79 (Jan. 2014), 204–223. <https://doi.org/10.1016/j.scico.2012.04.007>
- [34] Leonel Merino, Alexandre Bergel, and Oscar Nierstrasz. 2018. Overcoming Issues of 3D Software Visualization through Immersive Augmented Reality. In *2018 IEEE Working Conference on Software Visualization (VISOFT)*. IEEE, Madrid, 54–64. <https://doi.org/10.1109/VISOFT.2018.00014>
- [35] Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. 2017. CityVR: Gameful Software Visualization. (2017), 5.
- [36] Roberto Minelli and Michele Lanza. 2013. SAMOA – A Visual Software Analytics Platform for Mobile Applications. In *2013 IEEE International Conference on Software Maintenance*. 476–479. <https://doi.org/10.1109/ICSM.2013.76> ISSN: 1063-6773.
- [37] Martin Misiak, Andreas Schreiber, Arnulph Fuhrmann, Sascha Zur, Doreen Seider, and Lisa Nafeie. 2018. IslandViz: A Tool for Visualizing Modular Software Systems in Virtual Reality. In *2018 IEEE Working Conference on Software Visualization (VISOFT)*. IEEE, Madrid, 112–116. <https://doi.org/10.1109/VISOFT.2018.00020>
- [38] David Moreno-Lumbreras, Jesus M Gonzalez-Barahona, and Andrea Villaverde. 2021. BabiaXR: Virtual Reality software data visualizations for the Web. (2021).
- [39] Michael Moser, Josef Pichler, Gunther Fleck, and Michael Witlatschil. 2015. RbG: A documentation generator for scientific and engineering software. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, Montreal, QC, Canada, 464–468. <https://doi.org/10.1109/SANER.2015.7081857>
- [40] Sugumaran Nallusamy, Meei Hao Hoo, and Farizuwana Akma Zulkifli. 2021. Controlled Experiment for Assessing the Contribution of Ontology Based Software Redocumentation Approach to Support Program Understanding. *Computing and Informatics* 40, 5 (2021), 1025–1055. https://doi.org/10.31577/cai_2021_5_1025
- [41] Roy Oberhauser and Carsten Leon. 2017. Virtual Reality Flythrough of Program Code Structures. In *Proceedings of the Virtual Reality International Conference - Laval Virtual 2017 on - VRIC '17*. ACM Press, Laval, France, 1–4. <https://doi.org/10.1145/3110292.3110303>
- [42] Vaclav Rajlich. 2000. Incremental Redocumentation Using the Web. *IEEE Software* 17, 5 (Sept. 2000), 102–106. <https://doi.org/10.1109/52.877875>
- [43] Andreas Schreiber, Lisa Nafeie, Artur Baranowski, Peter Seipel, and Martin Misiak. 2019. Visualization of Software Architectures in Virtual Reality and Augmented Reality. In *2019 IEEE Aerospace Conference*. IEEE, Big Sky, MT, USA, 1–12. <https://doi.org/10.1109/AERO.2019.8742198>
- [44] B. Shneiderman. 1996. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE Comput. Soc. Press, Boulder, CO, USA, 336–343. <https://doi.org/10.1109/VL.1996.545307>
- [45] Harry Sneed and Chris Verhoef. 2019. Re-implementing a legacy system. *Journal of Systems and Software* 155 (Sept. 2019), 162–184. <https://doi.org/10.1016/j.jss.2019.05.012>
- [46] Frank Steinbrückner and Claus Lewerentz. 2013. Understanding software evolution with software cities. *Information Visualization* 12, 2 (April 2013), 200–216. <https://doi.org/10.1177/1473871612438785>
- [47] Scott Tilley. 1998. *A Reverse-Engineering Environment Framework*. Technical Report. Defense Technical Information Center, Fort Belvoir, VA. <https://doi.org/10.21236/ADA343688>
- [48] Juraj Vincur, Pavol Navrat, and Ivan Polasek. 2017. VR City: Software Analysis in Virtual Reality Environment. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, Prague, Czech

- Republic, 509–516. <https://doi.org/10.1109/QRS-C.2017.88>
- [49] Richard Wetzel and Michele Lanza. 2008. CodeCity: 3D visualization of large-scale software. In *Companion of the 13th international conference on Software engineering - ICSE Companion '08*. ACM Press, Leipzig, Germany, 921. <https://doi.org/10.1145/1370175.1370188>
- [50] Richard Wetzel, Michele Lanza, and Romain Robbes. 2011. Software systems as cities: a controlled experiment. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. ACM Press, Waikiki, Honolulu, HI, USA, 551. <https://doi.org/10.1145/1985793.1985868>
- [51] Sandra Yin and Julia McCreary. 1992. Myths and realities: Defining re-engineering for a large organization. In *NASA. Goddard Space Flight Center, Proceedings of the Seventeenth Annual Software Engineering Workshop*.
- [52] P. Young and M. Munro. 1998. Visualising software in virtual reality. In *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242)*. IEEE Comput. Soc, Ischia, Italy, 19–26. <https://doi.org/10.1109/WPC.1998.693276>