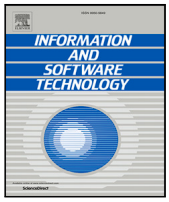
Information and Software Technology 151 (2022) 107007

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Collaborative program comprehension via software visualization in extended reality

Alexander Krause-Glau *, Malte Hansen, Wilhelm Hasselbring

Department of Computer Science, Kiel University, Christian-Albrechts-Platz 4, 24211 Kiel, Germany

ARTICLE INFO

Keywords:
Program comprehension
Software visualization
City metaphor
Extended reality
Virtual reality
Augmented reality

ABSTRACT

Context: In software visualization research, various approaches strive to create immersive environments by employing extended reality devices. In that context, only few research has been conducted on the effect of collaborative, i.e., multi-user, extended reality environments.
Objective: We present our journey toward a web-based approach to enable (location-independent) collaborative program comprehension using desktop, virtual reality, and mobile augmented reality devices.
Method: We designed and implemented three multi-user modes in our web-based live trace visualization tool ExplorViz. Users can employ desktop, mobile, and virtual reality devices to collaboratively explore software visualizations. We conducted two preliminary user studies in which subjects evaluated our VR and AR modes after solving common program comprehension tasks.
Results: The VR and AR environments can be suitable for collaborative work in the context of program comprehension. The analyzed feedback revealed problems regarding the usability, e.g., readability of visualized entities and performance issues. Nonetheless, our approach can be seen as a blueprint for other researchers to replicate or build upon these modes and results.
Conclusions: ExplorViz's multi-user modes are our approach to enable heterogeneous collaborative software visualizations. The preliminary results indicate the need for more research regarding effectiveness, usability, and acceptance. Unlike related work, we approach the latter by introducing a multi-user augmented reality environment for software visualizations based on off-the-shelf mobile devices.

1. Introduction

Visualizations are an established way to depict information, since they prove to be versatile in their application. As a result, software visualizations (SV) are used for different visualization concerns in software engineering. Their applications range from visualizing results of static code analysis [1] to rendering dynamics such as a software systems' runtime behavior [2] and also the evolution of software systems [3,4]. As a result, the visual abstraction of complex structures and behavior has potential to enhance the development and maintenance process of software.

A requirement for this overall achievement is a tool's capability to support the developers' comprehension of software, therefore decreasing the cognitive load during the recurring learning process [5,6]. While this task requires around half of the developers' time [7] and did not significantly change in the last three decades [8], professional developers still tend to use text-based tools such as integrated development environments, web browsers, and document editors to facilitate the comprehension [8,9]. However, the research community has developed different visualizations and tools to approach program comprehension. These differ in their scope, presentation, and devices, but overall examine how program comprehension can be facilitated via SVs.

Recently, SV approaches employ extended reality (XR) devices complementary to equipment that is common in workspace or learning environments. XR, i.e., the umbrella term for virtual (VR), augmented (AR), and mixed reality, strives to provide more immersive experiences. These devices often come as combination of head-mounted displays and controllers. Equipped with these cutting-edge input and output devices, users are able to experience and interact with the depicted content in new ways. Therefore, three-dimensional SVs can be seen as a good fit for XR, despite the fact that there are varying results regarding the effectiveness of XR for program comprehension [10–14]. Obviously, this effectiveness depends on the usability of an XR environment.

While interactivity and the resulting usability of SVs are crucial to realize benefits for program comprehension, both in non-XR and XR, the *collaboration* with other developers also can improve program

* Corresponding author.
E-mail address: akr@informatik.uni-kiel.de (A. Krause-Glau).

https://doi.org/10.1016/j.infsof.2022.107007
Received 17 December 2021; Received in revised form 18 May 2022; Accepted 8 July 2022
Available online 14 July 2022

comprehension and enable new use case scenarios. Face-to-face communication [15], collaborative development tools [16], and techniques like pair programming [17] show promising results to facilitate program comprehension. This can also be seen for SVs [18–21], although there exist so far only few research works in this area.

In this paper, we present the design and implementation of collaboratively explorable SVs. Users can choose conventional mouse interactions, common VR devices, or mobile devices to explore SVs via standard monitors, VR, or AR, respectively. To enable this range of interactions, we developed three complementary modes in our web-based SV tool ExplorViz. Each mode is session-based, independent of the collaborators' locations, and introduces different features for program comprehension. The collaboratively usable desktop mode of ExplorViz introduces a presenter mode, such that one person can be in charge of the visualization's interaction, e.g., rotation and zoom level. Other session members' visualizations are then synchronized, therefore everybody sees the same depiction, but can individually adjust details in the software visualization. In VR, multiple users share the same virtual space and are visible as avatars for other collaborators. Here, visualization elements are freely placeable while being synchronized among all users. In AR, using off-the-shelf mobile devices enables us to provide ExplorViz's AR visualization without the need for expensive equipment. A fiducial marker-based approach is used to visualize software systems in AR. Users can also use the same fiducial markers next to each other in meetings, therefore enabling a new approach to discuss about the visualized software. For all modes, we provide a supplementary package that includes videos and images showcasing each mode in practice [22].

We address the following research questions:

- Is a collaborative VR mode useful in the context of program comprehension?
- Is a collaborative AR mode useful in the context of program comprehension?

We conducted two preliminary user studies to evaluate our VR and AR implementations. Participants emphasized that it is sometimes difficult to read text labels and that the performance on weak mobile devices is insufficient. However, the XR environments indicate a positive effect on collaboration in the context of program comprehension.

Besides addressing these research questions, we report on our journey toward the web-based approach to enable (location-independent) collaborative program comprehension using both non-XR and XR devices, to impart the gained knowledge and our consequent design decisions.

The remainder of this paper is structured as follows. Section 2 presents related work in the context of collaboratively exploring software visualizations. As with the work in this paper, the related work is also categorized by non-XR, VR, and AR. We then introduce the method, the software architecture, and the basic visualizations of our web-based tool ExplorViz in Section 3. Section 4 depicts the main contribution of this paper, i.e., the design, implementation, and evaluation of our collaboratively usable software visualization modes in ExplorViz. We conclude this paper and present future work in Section 5.

2. Related work

In the following, we take a look at software visualization tools and their approaches for collaboration. We first discuss non-XR collaborative approaches and then proceed with examples in the realm of VR and AR.

2.1. Non-XR collaboration

SourceVis [18] is a collaborative software visualization tool. It focuses on usage scenarios with multiple co-located users who gather around a horizontally oriented multi-touch table. SourceVis offers 13 distinct visualizations in the categories exploration, structure, and evolution. Among these are the Metrics Explorer, Class Dependency View, and System Evolution View.

The different visualizations of SourceVis are designed to be used alongside one another. Therefore, the visualizations are displayed in movable, resizable, and rotatable windows which might overlap with other visualizations. MT4J[1] is employed to enable multi-touch functionalities but turns out to be time consuming for the implementation.

The results of a small qualitative user study imply that multiple users should be able to use the software visualizations simultaneously and that menus need to be easily accessible. In addition, the multiple visualizations are helpful to investigate different aspects of a software system but also can be challenging to gasp and switch between for novice users.

As opposed to several visualizations next to each other, ExplorViz provides a unified 3D software visualization. A variety of additional information can be displayed in menus on demand or through a heat map overlay [23]. ExplorViz also focuses to support collaboration in distributed as well as co-located scenarios.

2.2. VR collaboration

Jung et al. developed a tool for the collaborative software visualization in VR [21]. The visualization follows the city metaphor and combines static and dynamic data. Therefore, in addition to the structural data in the form of houses and districts, trace data is displayed as arcs which span from one house to another.

To increase the immersion for users, VR hardware such as the HTC Vive or Oculus Rift is employed. In contrast to ExplorViz, users cannot stand still and manipulate the visualization as wished but need to move or teleport to the desired part of the visualization. This design decision includes that the size of the user can be scaled instead of the visualization to get an overview.

Jung et al. conducted a controlled experiment to compare their approach to a traditional visualization using desktop computers. The collaboration during the study is limited to the interaction between a proband and an instructor. Thus, it remains unclear how well the collaboration between equitable users is supported.

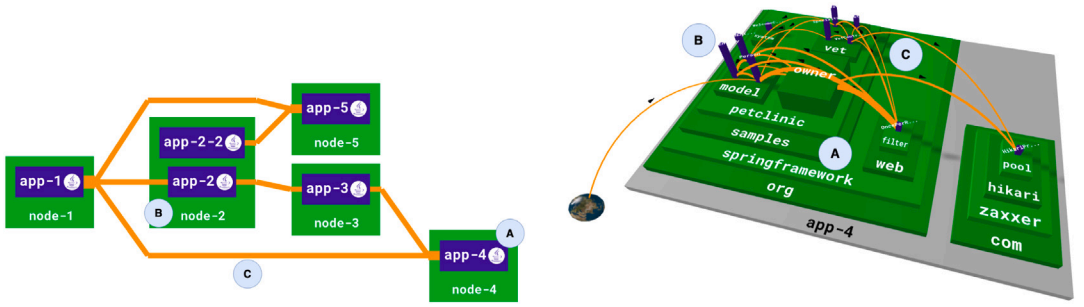
Overall, Jung et al. implemented a visualization approach which bears many similarities with the VR visualization approach of ExplorViz [10]. However, ExplorViz has a greater focus on offering many collaborative features as well as options to manipulate and customize the visualization.

2.3. AR collaboration

Henrysson et al. developed an early prototype for collaborative AR on mobile phones [24]. They used visual markers and Nokia mobile phones which possessed low-resolution cameras and screens. Even though the employed hardware exhibited low performance, they were able to implement a collaborative AR tennis game. The results of a conducted study indicate that the use of AR on mobile devices can enhance collaboration. ExplorViz builds upon a similar hardware setup but with modern mobile devices to enable collaborative exploration of software visualizations in AR.

In the field of software visualizations, IslandViz [25] is a tool which provides visualizations for both VR and AR. For the purpose of this subsection, we focus on its AR visualization.

[1] https://github.com/mschoettle/mt4j.



(a) Landscape perspective (with default color scheme) visualizing machines (green-colored), e.g., servers, their monitored applications (blue-colored), and distributed traces (orange-colored). The communication flow is unidirectional, i.e., runs from left to right.

(b) Application perspective (with default color scheme) visualizing an incoming distributed trace (globe) and method calls (orange-colored) between classes (blue-colored)

Fig. 1. ExplorViz' visualization perspectives. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

As the name suggests, IslandViz employs a custom metaphor to visualize component-based software architectures. For example, the overall software system is represented as an ocean containing several islands representing bundles (applications).

To realize the visualization in AR, IslandViz employs the Microsoft HoloLens as a hardware solution. The interaction with IslandViz is realized through gestures, voice commands, and gaze actions. Exemplary, an "Air-Tap" can be performed to select bundles and bring up additional information.

The use of Unity3D[2] in combination with the Holo Lens specific Mixed Reality Toolkit[3] allows for easy sharing of application states over multiple devices. However, there are no results for the collaborative use of IslandViz available.

Still, IslandViz is a tool that enables users to collaboratively explore software visualizations. In contrast to ExplorViz, the visualization approach and required hardware, as well as the employed software technologies, differ. As opposed to a monolithic model, we provide users with a landscape model, which gives an overview of the software system, and application models which are suitable for a more detailed software exploration. These models can be explored independently from each other as they can be placed on different markers. Furthermore, IslandViz uses specialized and expensive hardware for AR whereas our approach uses widely available and affordable devices.

3. ExplorViz

ExplorViz is our web-based tool for researching software visualizations [26,27]. Its development commenced in 2012 as open-source software and has gone through multiple architectural changes [28]. While ExplorViz started as a monolithic application, it is now developed as modular cloud-native software, following the twelve-factor app methodology.[4] Our current focus is to research a Software as a Service visualization approach for collaborative program comprehension. Therefore, we utilize technologies like container orchestration and stream processing to scale with varying workload. Most of ExplorViz' components are written in Java using the Quarkus microservice framework.[5] ExplorViz provides a dynamic software analysis approach as source for the visualization. However, the combination of static and dynamic analysis is beneficial and recommended [29] and will be supported by default in future versions of ExplorViz.

[2] https://unity.com/.
[3] https://github.com/microsoft/MixedRealityToolkit-Unity.
[4] https://12factor.net.
[5] https://quarkus.io.

3.1. Fundamentals

To understand ExplorViz, its user-centered focus, and the multi-user modes, one needs to understand the fundamentals of our approach in terms of data design and functionality. Since software is nowadays often distributed, users might need or generally want to monitor multiple distributed applications to collect the overall data for a software system. We address these requirements by introducing so-called *landscape tokens* (LT). LTs are unique identifiers that can be generated by users in the web-based frontend of ExplorViz. They are mandatory, since users must provide a valid LT in the instrumentation configuration for the monitoring. Monitored applications that use the same LT, form the overarching software landscape. This landscape is rendered by means of a top-level visualization that contains the pooled applications, the communication among each other, and more (see Section 3.2). Furthermore, the landscape visualization acts as entry point to more detailed visualizations of the contained applications. Users can always generate and use multiple LTs, therefore constructing different landscapes, e.g., two landscapes containing the same applications, but running on different machines. A LT and the data behind it belong to a single user. Thus, removing a LT also deletes its data.

3.2. On-screen 3D visualization

ExplorViz is developed with a web-based focus such that users can employ off-the-shelf devices with common web browsers. As a result, users can open the deployed Frontend to explore their monitored software systems within our web application. For that purpose, the ExplorViz Frontend component provides two visualizations for its users (Fig. 1). Fig. 1(a) depicts ExplorViz' landscape perspective. This visualization is inspired by UML deployment diagrams. It is used to visualize a software landscape, i.e., one or multiple applications which share the same LT (see Section 3.1). In the default color scheme, applications are rendered as blue boxes (Fig. 1(a)A). The green boxes represent the machines on which the applications are executed (Fig. 1(a)B). Distributed traces between multiple applications are visualized by means of orange communication lines (Fig. 1(a)C). These lines aggregate all traces from a source to its target in the currently visualized snapshot (see Section 3.1). The timeline at the bottom can be used to switch to a different snapshot, hence to see the overall landscape's execution at another point in time (not depicted). To see which method calls are included in those traces, therefore executed, users can proceed to open the second visualization.

Fig. 1(b) presents ExplorViz' application perspective. It can be accessed by clicking on an application within the landscape perspective (Fig. 1(a)A). We use the city metaphor [30] to visualize a single application as a three-dimensional software city. In our case, districts represent source code packages which can be interactively opened

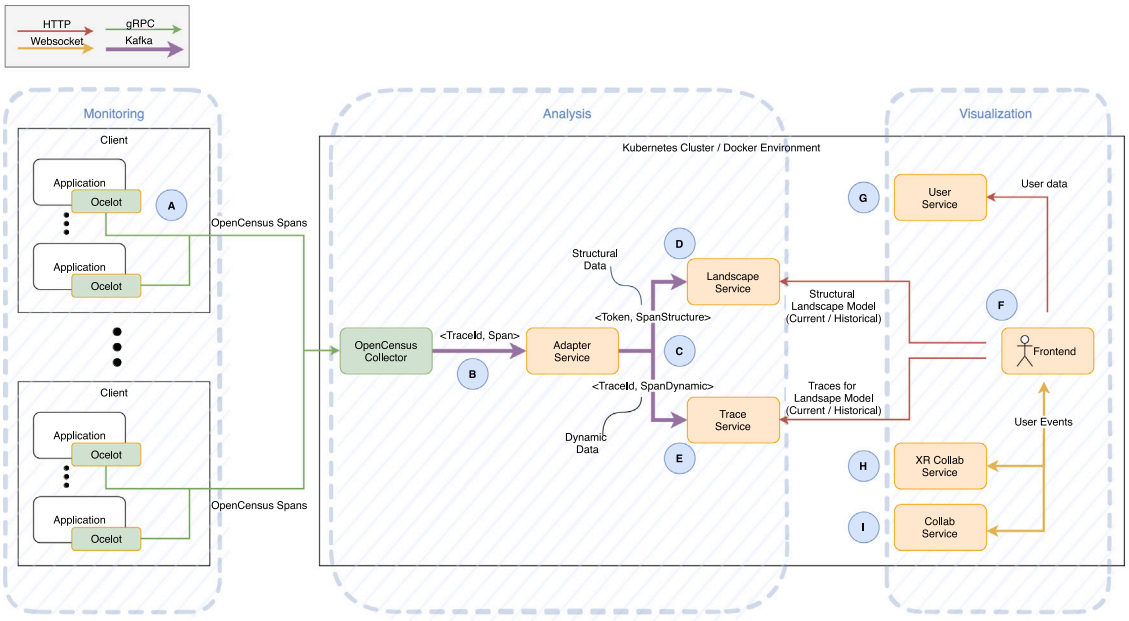


Fig. 2. (Simplified) ExplorViz architecture showing the involved applications/services (round-shaped boxes), their encompassing domains (blue-striped areas), and the resulting communication flow (arrows).

or closed, hence showing their internals (Fig. 1(b)A). Buildings are used to visualize classes which have been used during the application's execution (Fig. 1(b)B). A single communication line represents a method call on one class that was executed by another class or their instances, respectively (Fig. 1(b)C). Traces are visualized by a set of orange communication lines and can be interactively explored in a retractable window (not depicted). Here, users can also adjust settings such as the used colors within the visualization or manage a collaborative session. This is further explained in Section 4.2. Popups show additional information for an entity when a mouse-over event is triggered. We also provide a complementary heat map mode that users can open to explore the runtime behavior in a different way [23].

3.3. Architecture

Fig. 2 depicts ExplorViz' architecture. For the sake of simplicity, databases, reverse proxies, and auxiliary software are not depicted. Externally developed applications are depicted as green boxes, whereas our implementations use orange boxes. We divide the functionality of ExplorViz into three domains (areas with blue-dashed lines). Each domain and its internal applications are self-contained, respectively, therefore interchangeable by an implementation with the same external interfaces. Complete domains or single applications of a domain can run in multiple hosting solutions, e.g., two Docker environments on separated machines. Therefore, the depicted deployment is only an example.

ExplorViz' Monitoring Domain handles the data collection of desired applications (left of Fig. 2). It is the source for the resulting visualizations. Currently, ExplorViz only supports dynamic analysis as source. For that, we employ monitoring agents which handle instrumentation and monitoring of the observed software. Since there are various programming languages a user's application can be written in, we chose the open-source OpenCensus[6] library for the data exchange with the subsequent Analysis Domain (middle of Fig. 2). We started with a support for Java applications. However, OpenCensus and its successor OpenTelemetry[7] are highly utilized by various monitoring agents for different programming languages.

For Java, the instrumentation and monitoring is handled by Novatec's inspectIT Ocelot[8] (left of Fig. 2). Ocelot is a Java agent that can record method calls during an application's execution based on byte-code manipulation. Furthermore, it assembles execution traces from these calls. Afterwards, the traces can be exported with ready-to-use or user-developed exporters. We use the provided OpenCensus exporter which forwards any assembled traces to our analysis domain. In OpenCensus, this means that a trace is exported by individually sending out all its method calls with associated meta information. These so-called *spans* contain a unique trace identifier which can then be used by a receiver to reconstruct the trace. Fig. 2A visualizes multiple clients, running several applications each equipped with Ocelot (the blue encircled A on the left in Fig. 2). Generating workload for these applications, for example by executing use cases, will trigger the described behavior in Ocelot. Subsequently, the traces are exported in the form of spans via gRPC by the Ocelot OpenCensus exporter. Each span contains additional tags, which include the mandatory LT (see Section 3.1) and, for example, the application name. All this information is then used by the analysis domain to create the data that is visualized.

ExplorViz' Analysis Domain receives the spans from the Monitoring Domain, i.e., from the Ocelot Java agent. The Analysis Domain mainly uses stream processing, based on Kafka Streams[9] to send, receive, and analyze data between applications of this domain. These applications are depicted in the Analysis Domain in Fig. 2. We see that the Analysis Domain comprises four applications. The OpenCensus Collector is an OpenCensus component.[10] In our case, it acts as gateway to the ExplorViz Analysis Domain. However, it also provides more complex methods such as annotating or filtering spans. The collector has ready-to-use exporters which send spans to desired receivers. We use the collector's Kafka receiver to forward spans from the Monitoring Domain to the ExplorViz adapter service. The adapter and the following services are built as microservices using the Quarkus framework.

Fig. 2 shows that these services mainly communicate via Apache Kafka, more precisely using Kafka Streams. Using Stream processing

[6] https://opencensus.io/.
[7] https://opentelemetry.io/.
[8] https://www.inspectit.rocks.
[9] https://kafka.apache.org/documentation/streams.
[10] https://opencensus.io/service/components/collector.

enables us to easily scale out the ExplorViz Analysis Domain or only some of its parts, to adapt to varying workloads. The Adapter Service validates each incoming span. For that, it rejects spans which do not contain a LT or contain a non-existent LT. Each incoming span (Fig. 2B) is then separated into structural and dynamic data for performance improvements. We define structural data as the fraction of a span that is often repeated during the applications' execution. Examples are the hostname, application name, and fully qualified operation name that spans always contain. This information is indeed repeated, when the same methods are continuously called during the applications' execution. With this decomposition approach, we are able to reduce the data volume that needs to be persisted, since we only need to persist one representative for each structural data entry. Dynamic data is defined as the timing information of a called method, therefore used to reconstruct the traces which consist of multiple method calls. Each instance contains a unique key that can also be found in the related structural data entry. This enables us to enrich the structural information with their timing information.

After the decomposition into the two types of data, performed by the Adapter Service (Fig. 2C), structural and dynamic information are further online processed and persisted by the Landscape Service (Fig. 2D) and the Trace Service (Fig. 2E), respectively. The latter aggregates the dynamic information of the last ten seconds to multiple instances of our trace representation. Furthermore, it performs a set of reduction techniques to decrease the data volume, e.g., remove redundant occurrences of the same trace and use a representative instead. The Landscape Service creates a tree representation based on the received structural data. This is a performance improvement for the following rendering pipeline in our Frontend (usually a web browser). Both, the Landscape and the Trace Services provide HTTP endpoints to obtain their data.

ExplorViz' Visualization Domain uses these endpoints to obtain the data and finally visualize it. The main component of this domain is the ExplorViz Frontend (Fig. 2F). It provides access to the software visualizations and also contains the newly developed multi-user modes. The ExplorViz Frontend runs inside of the users' web browsers and is served by a web server. The Frontend updates the currently rendered software visualization every few seconds. This is based on the LT of the structural and dynamic data that was initially generated by the user in the Frontend component and used in the instrumentation configuration (see Section 3.1). LTs are managed by the User Service of the Visualization domain (Fig. 2G). This service also propagates 'create' or 'delete' events of LTs to the Adapter Service, where these events are used to validate incoming spans. The Frontend creates a snapshot that includes all traces and potential new structural information of these past seconds. This enables users to go back to a previous snapshot such that they can go back to the point in time where they stopped exploring the visualization. Therefore, no runtime visualization is lost. The remaining applications of the Visualization Domain are the multi-user services (Fig. 2H and Fig. 2I). These are explained in the upcoming Section 4.

3.4. Envisioned usage scenarios for future research

This paper reports on the first steps of our journey toward a platform that provides its users with powerful tools to collaboratively explore SVs. With this approach, we research if and how collaboratively usable SVs are useful in the context of program comprehension. While VR and AR devices are still in an early stage of development, we can nonetheless envision further potential usage scenarios for our approach. Some of the following features are work in progress or depend on the availability of specific hardware, such as commercially available AR glasses. However, we intend to outline our ideas, so that readers get an overview of the potential that might come with collaborative program comprehension via SV in the future.

Ubiquitous software visualization. Depending on the collaboration activity, working together usually requires some prior setup to truly emerge, e.g., a set of prepared hardware and software tools or an environment where the collaboration takes place. Our approach simplifies this setup with a software-as-a-service application that can be hosted in public or private cloud environments. Users do not need to manually setup a software stack, but can use the hosted SV application instead. Visualization data is persisted, such that you do not need to reproduce the data collection. Instead, users can traverse through time within our application to comprehend their recorded data and visualizations. This might increase the acceptance and utilization of SVs, so that they become an alternative to text-based tools [8,9] in the comprehension task. Furthermore, we might reconsider SVs in the future as a common tool in our development processes. For example, data collection could be automatically triggered from within a continuous integration pipeline and would not require a manual procedure.

Collaborative program comprehension via SV. A shared, ubiquitous, and collaboratively usable SV environment has the potential to introduce an enhancement for educational purposes. Instead of laboriously comprehending source code and external documentation directly, new developers might use a prepared SV that guides them through selectable use cases and the corresponding applications, source code packages, classes, and method calls. The visualization details might come with additional information such as comments or voice recordings. Users might select their preferred device, e.g., desktop, VR, or AR, for this education and interactively invite a supervisor if questions arise.

Collaboration with heterogeneous devices. Another potential scenario is the usage of AR glasses in (face-to-face) meetings. Developers might collaboratively analyze the structure and behavior of their application for a specific use case. Remote developers could then also join this meeting with their desktop computer or VR equipment.

4. Collaborative program comprehension

We examine collaborative program comprehension in different ways. Each approach was designed with specific usage scenarios in mind. Therefore, they were built with an independent focus on a specific feature or purpose, but are ultimately used to research how we can explore software in teams with different media. So far, we do not include a voice chat directly in ExplorViz. However, externally provided software is easy to obtain and use. Furthermore, our modes for collaborative program comprehension currently use the same SVs that ExplorViz' on-screen mode uses. We might introduce new visualizations in the future. However, the city metaphor for instance is an often used SV for both static [31] and dynamic [32] properties of software systems and proves to be a adequate visualization approach [33].

In Section 3.2, we already introduced ExplorViz' basic on-screen 3D visualization, which has originally been designed as a single-user tool for dynamic analysis employing the 3D city metaphor [34]. Based on this 3D visualization, we conducted a controlled experiment with physical 3D-printed software cities. We recap the experience with these 'physical' visualizations in Section 4.1. Section 4.2 explains how we used the observations of the 3D print approach to draft our multi-user mode. In parallel to printing 3D objects, we also conducted a preliminary user study for collaborative program comprehension with virtual reality devices, as will be reported in Section 4.3. Inspired by these experiences with physical objects and virtual reality devices, we devised an approach for collaborative program comprehension with augmented reality techniques, see Section 4.4.

4.1. Physical 3D-printed software cities

Our first work in the context of collaborative program comprehension examined the use of 3D-printed software cities [35], inspired by 3D printing in the manufacturing industry. We extended ExplorViz'

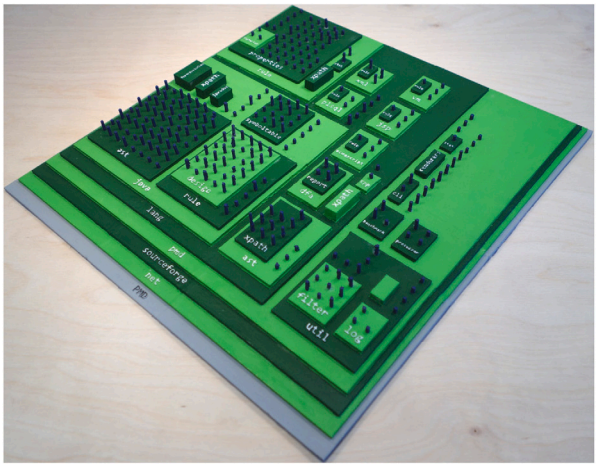


Fig. 3. 3D-printed software city (334 mm wide and 354 mm deep) [35].

with the necessary functionality to export OpenSCAD[11] files based on the monitored and analyzed runtime data from applications. Of course, such a printed 3D model of software does not visualize the dynamic behavior, but can only show a snapshot of the runtime behavior. Therefore, the focus of this approach was to compare the 3D print with its on-screen counterpart.

Fig. 3 depicts a 3D print showing a runtime snapshot of the source code analyzer PMD.[12] We see that most of the source code packages are opened, hence showing their internals, i.e., sub-packages and classes. Furthermore, method calls have been removed. In the on-screen version of ExplorViz, these are visualized with communication lines between the classes (orange lines in the previous Fig. 1). Their removal is due to the fact that we only visualize one single snapshot with the 3D print. Therefore, we focus on the structural entities of the runtime snapshot.

We conducted a controlled experiment to compare the 3D print with its on-screen counterpart in the context of program comprehension. In the following, we will summarize this evaluation, since it constitutes one step in our journey toward a web-based approach to enable collaborative program comprehension. We would like to point the reader to the related conference paper [35] and the published supplementary package [36] that contains all collected data including the 112 recordings of the participant sessions.

The experiment included 112 computer science students which were assigned in pairs to the control or experimental groups. The participation was voluntary and no compensation was received. Each group had to solve the same five tasks. We recorded the time spent for each task and the groups' gesture interaction (3D print) or their screen. After the experiment, we analyzed the correctness of the solutions. While the 3D print group was slightly faster and more correct in two tasks respectively, the overall results did not show significant findings on the time spent or the correctness of solutions.

The video recordings of our controlled experiment show interesting interactions among the probands of the 3D print groups [36]. Both members of a group often used their fingers to point on a specific detail of the printed software city. Overall, the interactions with the 3D print were frequently executed by both members. The desktop groups however used only one mouse cursor to point on details. It was of course operated by a single member of a group. As a result, some desktop groups reported that it occasionally was difficult to follow the mouse cursor which was controlled by the other group member.

Despite its impairment for the collaboration due to the single-user controls with the on-screen mode, we nevertheless see more potential in virtual environments to facilitate collaborative program comprehension. These environments are far more customizable, easier to access, and can incorporate the dynamic runtime behavior of the visualized software system. We expect that a collaboratively usable environment for SV exploration promotes the effectiveness of the visualization.

Another example that also profits from a collaborative use is pair programming. While this development technique started by using a personal computer to write and comprehend source code in teams of two, it nowadays can also be used remotely. For that, developers employ real-time collaboration tools to achieve a similar setup that is comparable to the initial idea of pair programming. The advantages include the location independence and the use of your own familiar development environment [37]. Furthermore, advantages that result from the initial idea of pair programming still apply, e.g., developers can share knowledge and collaboratively device how to develop [38].

4.2. On-screen multi-user mode

Equipped with the observations of the 3D print approach, we derived requirements and useful features for a multi-user mode (collaborative on-screen mode) in ExplorViz.

Design. ExplorViz' single-user on-screen mode (see Section 3.2) uses web technologies to simplify its use. If deployed, users only need to use their standard web browser to access a SV. We expect that a collaborative mode that is based on an already convenient to use counterpart should also incorporate a similar ease of use. However, multi-user modes come with different requirements in terms of their applicability and resilience, e.g., convenient connection establishment and disconnections, respectively. For ExplorViz' collaborative mode, we decided to realize a session-based connection pool. With that, users are able to host, join, or leave a collaborative session. Furthermore, simultaneous explorations are possible, since users might want to explore different software landscapes and their included applications.

As the name implies, real-time collaboration tools broadcast events of different users almost immediately. This affects the visibility of collaborators and their actions, since each user will see what the others do. In the context of ExplorViz, we identified that a user's mouse cursor is the primary object that is required to be broadcasted to other collaborators. It is used as pointing device to highlight details in a depicted visualization. However, the rotation of the software city, its status in form of opened and closed packages, selected entities, and the camera's zoom level also need to be considered for a collaborative environment. Some of these properties may easily be realized by using multiple colors, i.e., a unique color for a user.

For example, we can implement the event of selecting a class by coloring this class for everyone in the color of the event's initiator.

[11] https://openscad.org/.
[12] https://pmd.github.io.

Fig. 4. On-screen multi-user mode (with vision impairment color scheme) showing a software city and the green-colored interaction pointer of a session participant. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Unfortunately, this approach will not work with complex interactions, which affect users' perspectives. Broadcasting these events would, for example, continuously change the users' rotation of the software city, hence impairing the usability and effectiveness. Therefore, we decided against a real-time broadcast of those events. However, our design considers that users can adopt another user's perspective. Ultimately, this results in the same outcome, but still disregards the mentioned impairing for the usability.

With pilot testing, we observed that the users often adopt a specific user's perspective upon request of that user. A reoccurring question during this process was whether all users succeeded to adopt the perspective, such that the discussion about a visualization detail could take place. This slow-paced procedure can work for small teams, but is cumbersome for many participants or if someone joins the discussion later on. To counteract this problem, our design includes a presenter mode, whereby remaining users follow the perspective of the presenter. This executive user can furthermore choose which events of the remaining users should be broadcasted, e.g., selected entities or mouse cursors, and pass on the presentation ability to someone else.

While the presentation ability can be forwarded, hence shared, the data behind the visualization must always belong to a single person. In our design, this is the initial presenter. Therefore, participants cannot further analyze the visualized runtime behavior, unless they reproduce the same execution for an identical application instance. We address this problem by allowing data owners to share or grant permission to fork a software visualization. Forking enables users to create a copy of a visualization. This is comparable to a deep copy of the underlying data, where users can enhance their clone with more traces. A suitable scenario is a user who intends to compile all use cases of an application with a cohesive data set. With this approach, the user only needs to expand the clone with the missing use cases.

Implementation. From the viewpoint of ExplorViz' users, our implementation seamlessly integrates in our web-based application (see Section 3.2). For that purpose, the existing user interface (UI) was non-intrusively extended. As a result, the new multi-user mode does not interrupt or distract users from a currently explored software visualization, but still can easily be accessed. For that, we introduced a multi-user mode panel that can be invoked by our context menu. This menu can be opened on standard computers with a right click, and on mobile devices with a long press. The multi-user mode panel allows users to host, join, or leave a collaborative session with such a single click.

Fig. 2I depicts the Collab Service that realizes the near real-time collaboration. The Collab Service is a Quarkus-powered Java application that uses web sockets to bidirectionally exchange data with connected clients. These clients are the web browsers of ExplorViz' users which run the Frontend, therefore connect to the Collab Service via a web socket. The web sockets provide the foundation to support the simultaneous exploration. It uses sessions that users can host, join, or leave by clicking a single button in the related Frontend's UI context. Client-side actions in a session such as selecting a visualized class are then forwarded to the Collab Service, which subsequently broadcasts this event to remaining session participants.

Fig. 4 shows the perspective of a session participant during a collaboratively explored software visualization. We see a rendered software application and the opened multi-user mode panel. In this panel, all participants are depicted with their OpenID Connect[13] user name (Fig. 4A).

Next to the user names, we see a color and an eye symbol (Fig. 4B). The color is randomly chosen and represents the used color for the broadcasted mouse cursor of this participant (Fig. 4C). The eye symbol is a button that applies the perspective of the related participant to the current visualization. This facilitates communication, since the depiction of a detail that is currently discussed depends on the current angle, rotation, and package status (opened or closed) of the visualization.

4.3. VR mode

With the release of modern VR devices, we started to work on a single-user VR mode to explore SVs in 2015 [10]. This mode used the Oculus Rift Development Kit 1 and the Microsoft Kinect v2 sensor. The latter is a camera-based gesture recognition device. It was used to interact with the SV and later on replaced with VR controllers. We conducted a preliminary user study with eleven computer science students to evaluate the usability of this VR approach. While the camera-based gesture recognition showed its drawback due to a fixed position in the real world, we nevertheless expected potential for a beneficial use of VR in the context of program comprehension. Therefore, we further refactored our VR mode and designed it for collaborative use.

[13] https://openid.net/connect.

Design. In our experience, employing VR means that users often struggle with their setup. For example, some users are impaired by the wiring that several VR devices still require. While there are wireless standalone alternatives that counteract this problem, not all of these devices provide the required performance. Wireless adapters, which convert wired devices into wireless ones, are also difficult to operate due to their specific requirements on computer hardware. A successful SV-related VR mode should support a wide range of head-mounted displays (HMD). Since users pose different demands on such VR devices, we can therefore enhance the usability by supporting a user's favorite headset. We argue that a better usability results in more interaction between users, since they feel more comfortable in VR when using their desired device. As a consequence, this might affect the collaboration while exploring SVs.

VR applications are developed in various rendering engines, therefore might have some unforeseen problems that only occur on some hardware platforms. Additionally, various approaches in SV research follow gaming practices and use gaming engines to render the depiction and drive the connected VR devices. These engines often impose high requirements on the executive machine, hence are not available for everyone. Thus, our design of a collaborative VR mode should not exclude users by presuming high performance gaming machines.

Section 4.2 introduced a user's mouse cursor as the primary object that is required to be propagated to other session participants in the on-screen multi-user mode. Here, the mouse cursors act as (simple) avatars of connected users. Using VR enables us to visualize users in a more natural way, since we are not bounded to two-dimensional screens that render three-dimensional objects with a two-dimensional input device, i.e, a computer mouse. Our avatar design follows common VR practices and depicts session participants in the form of a rendered VR headset and its controllers. Based on pilot testing in previous versions of the VR mode, we currently decided against the use of more complex avatars. Human-like virtual avatars often appear as comic-like figures, if not carefully designed. For example, the animation of joints and limbs sometimes looks odd, which is not appropriate for a professional environment. However, different virtual avatars may promote the recognition of session participants and therefore might facilitate interaction between users. For example, if a user wants to discuss a visualization detail, we could easier find this person in the virtual environment and consequently recognize the detail that this person is concerned about. Our design addresses this problem in a different way by introducing a *ping* feature. This helps users to find the visualization detail that another user wants to highlight by means of attaching a fluctuating orb to the detail. Arrows appear in the field of view of a user, if this orb is not in her or his viewing frustum. They indicate the direction that the user needs to face to find the fluctuating orb. Additionally, each controller is equipped with a virtual laser beam that is used to select or open and to ping visualization details. This beam is also visible to the other session participants and once more helps to find a visualization detail that other users refer to.

Our design not only deals with the collaborative exploration of SVs, but also with the way users can collaboratively build the surrounding environment. Therefore, it has a different focus on collaboration than the on-screen multi-user mode (see Fig. 4). ExplorViz' users share the same space in VR. We utilize this property such that users are allowed to open multiple applications of the underlying landscape (see Section 3.1) and explore their chosen application in a free area of the shared environment. Other session participants can freely walk and teleport in the shared environment, therefore seamlessly join a discussion of a different application. Additionally, every user can freely stick information windows anywhere in the shared space. These are similar to the on-screen versions, but are not limited in their number and position. This allows users to easily share information, while independently exploring the visualization.

Implementation. A collaboratively usable VR mode for SVs should not exclude users by requiring highly performant gaming machines or the use of specific devices. Our implementation addresses this by utilizing the Three.js[14] library and the WebXR API[15] as rendering engine and XR driver, respectively. Due to their origin as web technologies, both are optimized for less performant devices. Three.js is already in use for ExplorViz' on-screen modes (see Sections 3.2 and 4.2), hence allowing us to rebuild a familiar visualization in VR. WebXR is an API that is included in recent versions of standard web browsers. It is natively supported by Three.js and allows the use of connected XR devices for web content. Users can employ outside-in tracking, e.g., full-size room-scale VR environments, and inside-out tracking that is for example used by the Oculus Rift S. Furthermore, standalone devices such as the Oculus Quest 2 or smartphones equipped with VR headsets also support WebXR. Overall, WebXR supports a vast amount of VR and AR headsets, therefore allowing users to use their favorite device. Equipped with these technologies, our VR mode can still run inside of a browser such as Mozilla Firefox or Google Chrome and does not require any further software. It is still part of ExplorViz' Frontend and can be used within the same deployed instance by means of a single click.

Fig. 2 depicts the XR Collab Service with the architecture of ExplorViz. This microservice provides the backend functionality to host, join, or leave a collaborative VR session. Furthermore, it acts as broadcaster to propagate events between clients that attend the same session. Overall, it is developed similar to its on-screen counterpart (see Section 3.2), but uses different events because of the three-dimensionality in VR.

Fig. 5 shows a screenshot of our collaborative VR mode from the viewpoint of a session user. We use 3D objects of VR HMDs and controllers to depict session participants (Fig. 5A). Examples for sticky information windows are shown in Fig. 5B. We also see an implementation of the ping feature. Fig. 5C depicts the fluctuating orb that is used by the same colored (red) session participant to highlight a detail in a visualized software city. The purple arrow points to another orb of a different participant (Fig. 5D). Users can adjust their settings such as their virtual height via a head-up menu (not depicted). This menu can also be used for session management or to select a different visualization snapshot (see Section 3.1). If selected, the latter may result in re-layouting of the underlying landscape visualization (Fig. 5F) and its currently opened software cities (Fig. 5G).

Evaluation. We conducted a pilot user study to evaluate the usability of our collaborative VR mode. In the following, we will summarize this evaluation, since it constitutes one step in our journey toward a web-based approach to enable collaborative program comprehension. We would like to point the reader to the related thesis [39] and the published supplementary package [40] that provides the results and the material to reproduce the study.

For that, 24 subjects were invited to test this mode. The participation was voluntary and no compensation was received. The twelve resulting teams of two were comprised of nine computer science students, eight computer science researchers or PhD candidates, and seven professional developers. Teams were freely chosen. Therefore, most of the team members already knew each other and worked together. We used different rooms to physically separate all participants and mimic a remote way of working. All subjects used a headset to communicate with their respective team member via an externally provided voice chat. Regarding VR devices, we employed the HTC Vive Pro (1440 x 1600 pixels per eye) and Oculus Rift (1080 x 1200 pixels per eye).

The user study started with a preparation phase that explained the to-be explored software system CoCoMe.[16] Furthermore, all subjects

[14] https://threejs.org.

[15] https://immersiveweb.dev.

[16] https://github.com/cocome-community-case-study/cocome-cloud-jee-platform-migration.

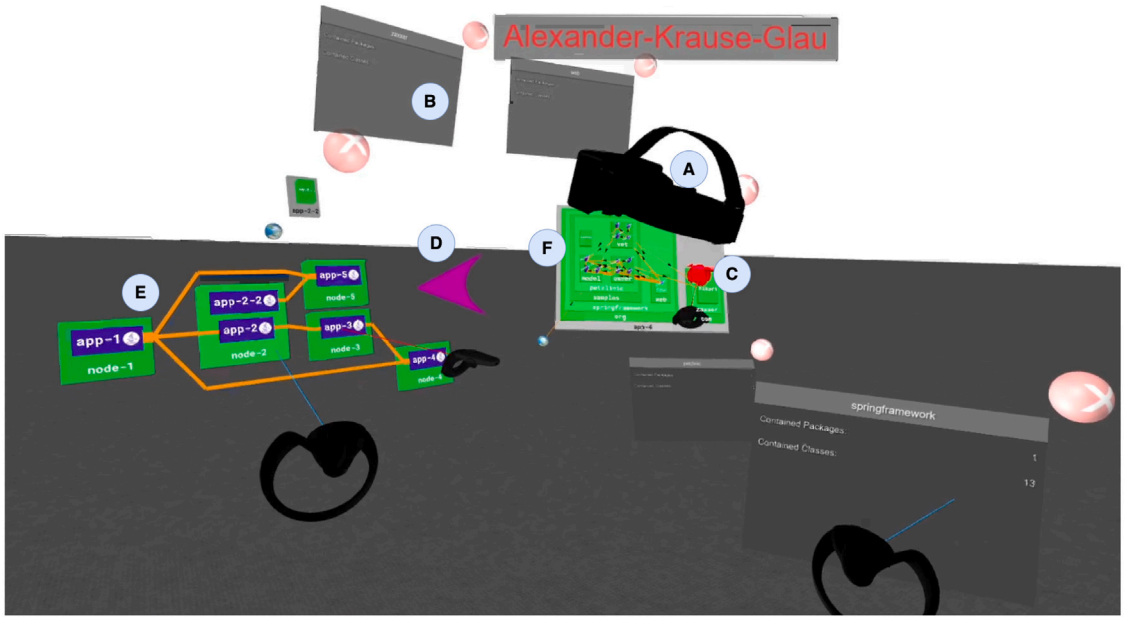


Fig. 5. Multi-user VR mode from the viewpoint of a session user. The purple arrow indicates that another session participant wants to highlight a detail that lies outside of the view. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

were asked to answer personal information by means of a questionnaire. After that, we continued with a training phase such that all subjects were able to familiarize theirselves with the VR environment. This included the recognition of the other session participant in VR as well as visualization and interaction features, e.g., using the laser beam or moving along in the VR space. In the subsequent assignment phase, all teams were asked to solve seven program comprehension tasks. These tasks were categorized in the form of structural and dynamical comprehension and were sorted by the task's difficulty. They were solved in the context of a single visualized snapshot of the runtime behavior (see Section 3.1) of CoCoMe. The debriefing phase was the final step of the user study. Here, we asked all subjects for feedback in terms of, e.g., the perceived task difficulty and the usefulness of collaboratively comprehending software in their opinion.

The results of the assignment and debriefing phase as well as our observations during the study's execution were used to answer our first research question, i.e., is a collaborative VR mode useful in the context of program comprehension? The provided feedback for the perceived usefulness of the collaboration aspect had to be answered for each task. The mean values indicate that the collaboration in VR was helpful. We noted a correlation between the usefulness and the difficulty of a task. This might indicate that increasing complexity promotes collaboration. However, as the time passed during the study, subjects might have become more experienced and familiar with the overall VR experience and their team member. Therefore, previous personal obstacles might have vanished as the time went by. Regardless of the reason, we conclude that our VR mode is useful to collaboratively comprehend software. This is supported by the high correctness of the given answers and the average agreement that our VR mode promotes communication as well as interaction. The latter activities were frequently observed by us in the assignment phase. For example, subjects used the selection feature to highlight a detail in the visualization. With the awareness of the other team member, which was also positively rated, this feature set seemingly facilitated the collaboration.

The results and observations indicate that our VR environment can be helpful for collaborative work in the context of program comprehension. However, more research is required to reliably answer our second research question. We used the preliminary VR user study to gather first qualitative feedback from a small set of probands. After a refinement phase based on the qualitative results, we will compare each mode based on quantitative results, e.g., via controlled experiments [41].

Threats to validity. One threat to validity that might have influenced our results is the number of subjects. However, 24 participants should be sufficient for a qualitative pilot user study. A higher number of participants might yield more reliable results.

Another potential threat are performance problems that were mentioned in the feedback phase. We expect that we can counteract this problem with a higher computing power and using the newest, potentially more powerful, devices. Our employed machines only used graphic cards that represent the minimum requirements for VR at that time.

We communicated orally with all subjects during the assignment phase to read out the tasks. Misunderstandings might have influenced the results. We expect that a textual task representation inside of the VR space could be more suitable, despite the fact that the conductors repeated the tasks if a subject did not hear it well enough.

4.4. AR mode

As described in the previous section, the integration of AR into ExplorViz is a consequent step to complement the existing collaborative modes and allow for a multitude of usage scenarios. Other visualization approaches use webcams or HMDs like the Microsoft HoloLens[17] to render software cities [12,42]. While Microsoft's HoloLens might provide good interaction techniques and immersion, it is quite expensive and not in widespread use. Therefore, we designed and implemented our collaborative AR mode around affordable commodity hardware, namely mobile devices with a focus on tablets. An example for the use of our AR mode is shown in Fig. 6.

Design. The AR visualization approach of ExplorViz is based upon multiple 3D models, i.e., models for the software landscape and applications. As opposed to a single, integrated 3D model, this allows the independent placement, scaling, and rotation of the different applications which belong to a software landscape. The use of multiple 3D models also reduces the visual size of a model and can improve performance since less relevant applications may remain hidden. Both of these aspects benefit mobile devices which exhibit a smaller screen size and less computational power than their desktop counterparts.

[17] https://www.microsoft.com/en-us/hololens.

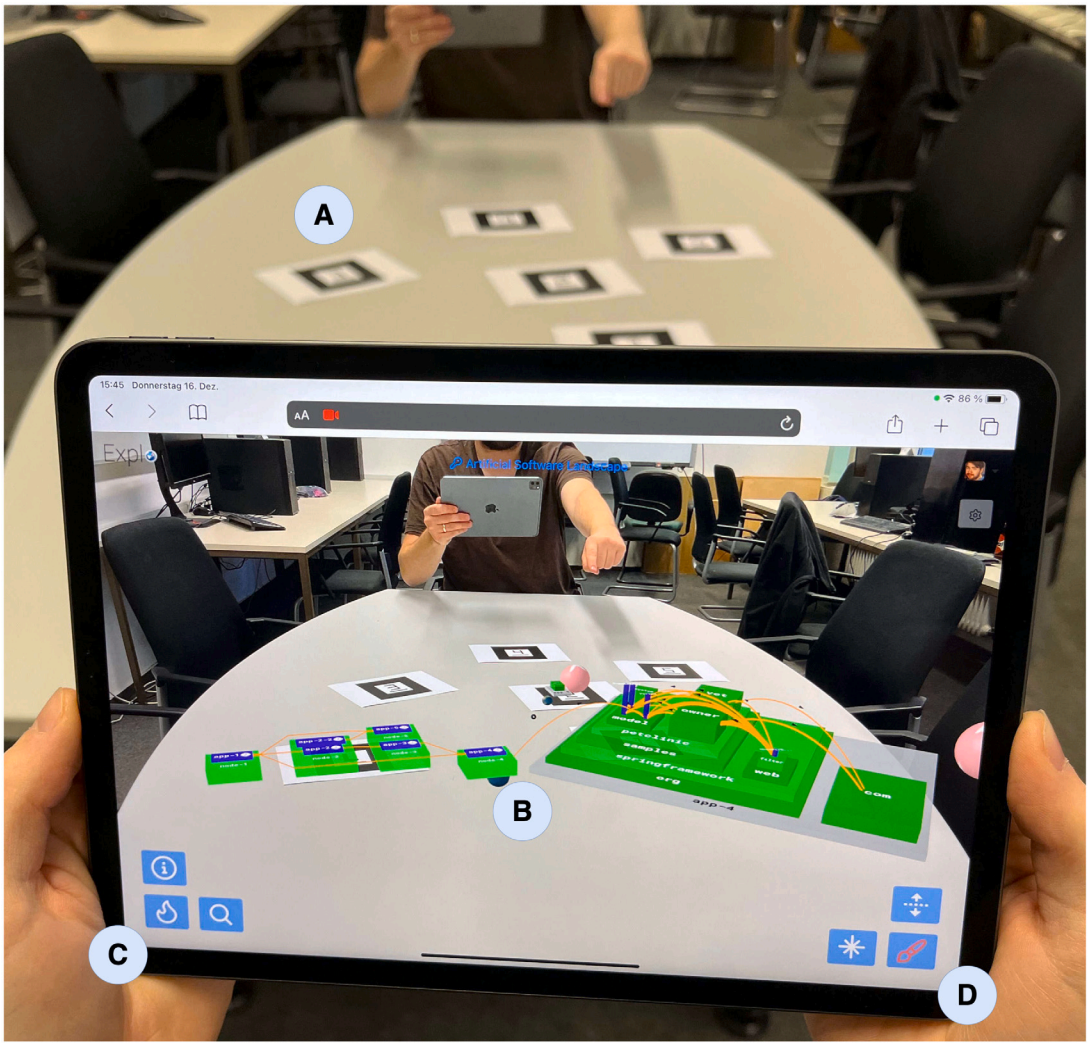


Fig. 6. Real-world scenario showing the multi-user AR mode. The augmented reality fiducial markers are used as reference points to place the software visualization in the real word.

In order to achieve AR, the virtual models need to be placed in accordance with the live video stream which is captured by the mobile devices. We decided to employ paper-printed fiducial markers (Fig. 7) as a point of reference for this purpose (Fig. 6A). A designated marker hosts the landscape model while other markers are numbered and can be associated with an application model upon its opening. The use of markers makes the placement of models predictable and does not require that the mobile devices compute an accurate 3D representation of their surroundings. The placement of models in the real world can easily be adapted by moving the paper markers instead of using touch gestures. Furthermore, the models can easily be referenced by the letter or number of the marker they are placed upon for collaborative usage scenarios.

The envisioned usage scenarios of the multi-user on-screen, VR and the AR modes differ significantly. For example, the AR mode could help to introduce software visualization as a tool during team meetings. Tablets are wireless and easy to carry, each participant can use an own device, and the interaction with models is aided by the use of paper-printed markers. Aside from tablets, we strive to support a great number of devices from smartphones to larger mobile computers with a camera and touchscreen. To enable diverse usage scenarios, our location-independent collaboration approach is also employed for the AR mode and allows for its use in hybrid or online formats.

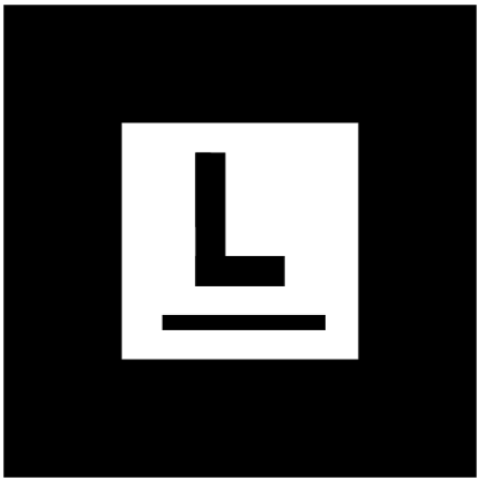


Fig. 7. Example for a paper-printed fiducial marker that is used as a point of reference for our AR approach.

Implementation. Analogous to the VR mode, users can easily switch back and forth between the on-screen visualization and the AR mode

via a context menu entry. The AR mode requires a web browser, access to a camera, and a secure connection in order to work. Therefore, a desktop computer with an attached webcam suffices to generate a live camera feed with AR elements. However, a realistic immersive visualization can only be accomplished with devices where the screen and camera face in opposite directions, such as tablets and smartphones.

To place the virtual 3D models accurately within the captured video footage, we employ AR.js.[18] AR.js is a Javascript library which provides AR features for web applications. AR.js supports the recognition of images, markers, or the use of location data to align objects accurately in video footage. In accordance with our previously mentioned design considerations, we chose to leverage the marker tracking capabilities of AR.js. In this way we can virtually place the landscape and application objects on markers and achieve the AR visualization of ExplorViz (Fig. 6B).

The design of the user interface (UI) is adjusted toward the two-handed use of tablets. Tablets provide high mobility, a moderately sized screen, and fulfill the requirements for the AR mode, i.e., they bring both a modern browser and camera. We argue that holding the tablet with two hands is preferable to avoid a shaky camera, occlusion of elements by touch inputs, and to reduce signs of fatigue caused by the device's weight. Therefore, necessary UI buttons are displayed in the corners of the device, such that they can be easily accessed via the users' thumbs. This is comparable to holding and interacting with a gamepad controller. In addition to tablets, we actively support the use of other mobile devices through dynamic sizing of UI elements and by providing plenty of options to customize the UI. In the style of some mobile video games, we put a crosshair-like element in the center of the screen as a point of reference. The thereby targeted elements can be interacted with by several buttons which are placed in the bottom corners of the screen. The buttons in the bottom left corner (Fig. 6C) can be actuated to display additional information, show ExplorViz's heat map overlay to easier identify runtime behavior changes [23], and toggle a virtual magnifier. The buttons in the bottom right corner (Fig. 6D) enable the opening/closing of packages, the colored highlighting of elements, and the temporary pinging on elements for collaborative use. The button's size and spacing are configurable such that they can be easily interacted with by the user's thumbs.

In addition to the buttons, users are enabled to use common (multi-)touch gestures for the view manipulation which are implemented with the help of Hammer.js.[19] A pan gesture enables a user to move a model horizontally on its marker, pinch gestures can be used to scale an object, and the rotation of two fingers results in a corresponding rotation of the targeted landscape or application object. The touch gestures require a user to temporarily hold the tablet with one hand. However, since the targeted objects are relatively large, these gestures require less precision than the manipulation of, e.g., a single class within an application.

To enable the collaborative use of the AR mode, we use the same Collab Service as the VR mode. Therefore, the session management is technically identical but received a dedicated UI which is optimized for touchscreen use. As opposed to the VR mode, not all properties of the depicted models are synchronized among the users of a session. The orientation and scaling of models can be configured independently per device as we address a heterogeneous set of devices and a multitude of usage scenarios. For example, the screen size and camera properties of mobile devices can differ significantly and cause different scalings of objects to be optimal. Still, the assignments of markers and applications and their general state, including opened packages and highlighted elements, remain synchronized. In addition to the highlighting of individual elements, users can temporarily ping any location within an application or landscape object to facilitate communication.

Evaluation. We conducted a preliminary user study to gather first impressions about the usability of our AR mode. In the following, we will summarize this evaluation, since it constitutes one step in our journey toward a web-based approach to enable collaborative program comprehension. We would like to point the reader to the related theses [43] and the published supplementary package [44] that contains the questions and results of the study.

20 subjects participated voluntarily and without compensation in the user study. Among the participants were seven computer science students, eight researchers, three software architects, and two software developers. The subjects conducted the user study in teams of two and mostly knew each other.

Due to the COVID-19 pandemic, the study was conducted in a remote setup, i.e., the subjects stayed at home or participated from their workplace. Therefore, the teams of two and an instructor used a video conferencing tool for audio communication during the experiment. Also, the distributed setup made it necessary for the subjects to use their own devices for the user study. As not all subjects owned a tablet, nine subjects used a smartphone instead. Furthermore, half of the subjects had no access to a working printer at home and reverted to displaying markers on a peripheral screen.

The course of the study is similar to the preliminary user study for the VR mode. A small example visualization was employed to familiarize the subjects with ExplorViz and its AR mode. Subsequently, BIMSWARM,[20] a web application in the domain of building information modeling, was introduced. BIMSWARM employs a microservice architecture and was still in development at the time of the user study. Professional software developers and architects of the BIMSWARM project participated as subjects in the evaluation. The introduction of BIMSWARM included the presentation of an exemplary registration of a user since we used a snapshot of the registration process for the software visualization during the following assignments. The assignment phase asked the subjects to collaboratively answer questions about both structural and dynamic aspects of the displayed snapshot. We also asked questions which required it to compare multiple microservices or asked to reason about the displayed data, thereby encouraging the subjects to communicate with each other.

The results of the assignment and debriefing phase as well as our observations during the study's execution were used to answer our second research question, i.e., is a collaborative AR mode useful in the context of program comprehension? In the final debriefing phase, the subjects were asked to fill out an online survey. Among other aspects, the survey asked the subjects to give feedback and rate the employed visualization approach, implemented features, and the collaborative work experience.

The visualization in general, e.g. concerning the overall layout, received good feedback. However, as to be expected, the visualization achieved better ratings on tablets than on smartphones. Especially for the readability of text and the distinguishability of communication lines, small displays bring disadvantages. The usability of the implemented features such as pinging or the opening and closing of packages received very good ratings. Regarding collaboration, the gathered feedback reveals that the collaborative use of ExplorViz is perceived as beneficial, can aid program comprehension, and that the developed AR mode is suitable for collaborative work in general. We also noted during the study that collaboration of subjects leads to more correct results since faulty propositions for questions of one team member often leads to discussions with the other team member and, without the need of the instructor to intervene, resulted in an correct answer.

The results and observations indicate that our AR environment can be helpful for collaborative work in the context of program comprehension. However, more research is required to reliably answer our second research question. We used the preliminary AR user study to gather first qualitative feedback from a small set of probands. After a refinement phase based on the qualitative results, we will compare each mode based on quantitative results, e.g., via controlled experiments [41].

[18] https://ar-js-org.github.io/AR.js-Docs.

[19] https://hammerjs.github.io.

[20] https://www.bimswarm.de

Threats to validity. One threat to validity concerns the employed equipment and environment of the subjects. As it was not possible for us to provide a controlled environment, many aspects such as the hardware specifications of the employed devices vastly differ from subject to subject. On the other hand, this heterogeneous setup helped to collect valuable and multifaceted feedback.

The limited number of participants yields another threat to validity. However, the 20 subjects gave plenty of feedback, which should be sufficient to guide the upcoming steps for the development of our AR mode. Nonetheless, in order to generate more accurate and quantitative results, a larger number of subjects would be required.

5. Conclusions and future work

In this paper, we presented our journey to enable collaborative program comprehension via SVs using both non-XR and XR devices, to impart the gained knowledge and our consequent design decisions. We introduced our designs and the resulting implementations for three multi-user modes in ExplorViz. All of these modes allow remote work, hence facilitate location-independent collaborative program comprehension. The multi-user on-screen mode allows users to collaboratively explore SVs on their personal computers and laptops. Similar to remote pair programming, this has the advantage that users work in their familiar and possibly customized working environments. In contrast, the VR and AR modes provide immersive environments that can be accessed with off-the-shelf devices. All modes are now available in our open-source, web-based live trace visualization tool ExplorViz. They seamlessly integrate in ExplorViz' existing UI and do not prevent users from exploring SVs due to complex configuration requirements. For all modes, we provide a supplementary package that includes videos and images showcasing each mode in practice [22]. Both, the VR and AR modes were evaluated in pilot studies. The evaluation results indicate that XR modes are useful to collaboratively comprehend software. However, we require more in-depth research to quantify the usefulness of each mode, e.g., via controlled experiments. Then, we will be able to reliably answer our research questions.

Regarding future work, we strive for a heterogeneous multi-user mode in which on-screen, VR, and AR users can collaboratively work together, both independent of the location or in the same room [45]. The isolation of users and the environmental change that come with putting on VR displays convinced some SV researchers to substitute VR with AR [46]. We do not consider AR as generally more suitable to explore SVs. In our experience, the used devices not only influence the effectiveness and efficiency of SVs [45], but also the acceptance of the SV itself. That is why we want to provide feature parity for all three modes first, so that we can eventually compare each mode based on quantitative results.

Exploring SVs can also be seen as an activity that is not directly connected to live coding, especially when we are concerned with dynamic runtime analysis. Users should be allowed to seamlessly switch the environment in the form of leaving their workstation to use AR or even full-size VR equipment. Also, they should be able to receive the same VR environment when using a standalone HMD that may readily be available to them.

ExplorViz' upcoming static code analysis support is planned to enable a live visualization of structural source code changes while coding. This envisioned approach is similar to the features of Elliott et al. [47] and [VR]IDE.[21] However, we do not plan rebuilding a full-fledged IDE in VR due to their overall complexity and customizability. In our approach, users will be able to setup a collaborative session that can be used to explore software changes while concurrently coding in their IDE. The executive user (presenter) can then simply use a standalone VR or AR headset and collaboratively explore the code changes with other session participants in an immersive environment. We are aware that some users do not want to use VR due to personal issues, e.g., motion sickness. Thus, the planned live-coding feature will also be available in ExplorViz' other modes. Overall, we argue that users should be able to choose which type of immersion and devices they prefer and use.

For collaboration, we are currently testing new and supportive features that promote collaborative program comprehension. For example, we plan on integrating a *Tag and Comment* feature that allows users to pin information to a visualization detail for the currently visualized snapshot. This is comparable to real world sticky notes. Session participants can leave a typed or voiced comment. Other users are able to see that note in their visualization. Furthermore, they are able to respond to it and for example provide valuable feedback to a question or comment.

[21] https://github.com/Vito217/VRIDE.

CRediT authorship contribution statement

Alexander Krause-Glau: Conceptualization, Methodology, Software, Supervision, Writing – original draft, Investigation, Visualization. **Malte Hansen:** Conceptualization, Methodology, Software, Investigation, Writing – original draft. **Wilhelm Hasselbring:** Conceptualization, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Kevin Lotz, Daniel König, Marcel Bader, Justin Andresen, and Johannes Brück for their contributions with implementing and evaluating some of the features presented in this paper.

References

[1] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer, 2007, http://dx.doi.org/10.1007/978-3-540-46505-8.

[2] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, R. Koschke, A systematic survey of program comprehension through dynamic analysis, IEEE Trans. Softw. Eng. 35 (5) (2009) 684–702, http://dx.doi.org/10.1109/TSE.2009.28.

[3] R.L. Novais, A. Torres, T.S. Mendes, M. Mendonça, N. Zazworka, Software evolution visualization: A systematic mapping study, Inf. Softw. Technol. 55 (11) (2013) 1860–1883, http://dx.doi.org/10.1016/j.infsof.2013.05.008.

[4] F. Pfahler, R. Minelli, C. Nagy, M. Lanza, Visualizing evolving software cities, in: Proceedings of the 8th IEEE Working Conference on Software Visualization, VISSOFT 2020, 2020, pp. 22–26, http://dx.doi.org/10.1109/VISSOFT51673.2020.00007.

[5] M. Yousoof, M.S. Baba, R. K., Performance evaluation of the software visualization tools and a new framework to manage cognitive load in computer program learning, WSEAS Trans. Inf. Sci. Appl. 5 (2008) 655–663.

[6] V. Winter, M. Friend, M. Matthews, B. Love, S. Vasireddy, Using visualization to reduce the cognitive load of threshold concepts in computer programming, in: Proceedings of the 49th IEEE Frontiers in Education Conference, FIE 2019, 2019, pp. 1–9, http://dx.doi.org/10.1109/FIE43999.2019.9028612.

[7] K. Bennett, V. Rajlich, N. Wilde, Software evolution and the staged model of the software lifecycle, Adv. Comput. 56 (2002) 1–54, http://dx.doi.org/10.1016/S0065-2458(02)80003-1.

[8] J. Siegmund, Program comprehension: Past, present, and future, in: Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering, Vol. 5, SANER 2016, 2016, pp. 13–20, http://dx.doi.org/10.1109/SANER.2016.35.

[9] X. Xia, L. Bao, D. Lo, Z. Xing, A.E. Hassan, S. Li, Measuring program comprehension: A large-scale field study with professionals, IEEE Trans. Softw. Eng. 44 (10) (2018) 951–976, http://dx.doi.org/10.1109/TSE.2017.2734091.

[10] F. Fittkau, A. Krause, W. Hasselbring, Exploring software cities in virtual reality, in: Proceedings of the 3rd IEEE Working Conference on Software Visualization, VISSOFT 2015, 2015, pp. 130–134, http://dx.doi.org/10.1109/VISSOFT.2015.7332423.

[11] R. Oberhauser, C. Lecon, Gamified virtual reality for program code structure comprehension, Int. J. Virtual Real. 17 (2) (2017) 79–88, http://dx.doi.org/10.20870/IJVR.2017.17.2.2894.

[12] L. Merino, A. Bergel, O. Nierstrasz, Overcoming issues of 3D software visualization through immersive augmented reality, in: Proceedings of the 6th IEEE Working Conference on Software Visualization, VISSOFT 2018, 2018, pp. 54–64, http://dx.doi.org/10.1109/VISSOFT.2018.00014.

[13] J. Dominic, B. Tubre, J. Houser, C. Ritter, D. Kunkel, P. Rodeghero, Program comprehension in virtual reality, in: Proceedings of the 28th International Conference on Program Comprehension, in: ICPC, vol. 20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 391–395, http://dx.doi.org/10.1145/3387904.3389287.

[14] D. Baum, S. Bechert, U. Eisenecker, I. Meichsner, R. Müller, Identifying usability issues of software analytics applications in immersive augmented reality, in: Proceedings of the 8th Working Conference on Software Visualization, VISSOFT 2020, 2020, pp. 100–104, http://dx.doi.org/10.1109/VISSOFT51673.2020.00015.

[15] T. Roehm, R. Tiarks, R. Koschke, W. Maalej, How do professional developers comprehend software? in: Proceedings of the 34th International Conference on Software Engineering, ICSE 2012, in: ICSE '12, IEEE Press, 2012, pp. 255–265.

[16] R. DeLine, M. Czerwinski, G. Robertson, Easing program comprehension by sharing navigation data, in: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2005, 2005, pp. 241–248, http://dx.doi.org/10.1109/VLHCC.2005.32.

[17] A. van Deursen, Program comprehension risks and opportunities in extreme programming, in: Proceedings of the 8th Working Conference on Reverse Engineering, WCRE 2001, 2001, pp. 176–185, http://dx.doi.org/10.1109/WCRE.2001.957822.

[18] C. Anslow, S. Marshall, J. Noble, R. Biddle, SourceVis: Collaborative software visualization for co-located environments, in: Proceedings of the 1st IEEE Working Conference on Software Visualization, VISSOFT 2013, 2013, pp. 1–10, http://dx.doi.org/10.1109/VISSOFT.2013.6650527.

[19] J. Dominic, B. Tubre, C. Ritter, J. Houser, C. Smith, P. Rodeghero, Remote pair programming in virtual reality, in: Proceedings of the 36th IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, 2020, pp. 406–417, http://dx.doi.org/10.1109/ICSME46990.2020.00046.

[20] R. Koschke, M. Steinbeck, SEE your clones with your teammates, in: Proceedings of the 15th IEEE International Workshop on Software Clones, IWSC 2021, 2021, pp. 15–21, http://dx.doi.org/10.1109/IWSC53727.2021.00009.

[21] F. Jung, V. Dashuber, M. Philippsen, Towards collaborative and dynamic software visualization in VR, in: J.B. Andreas Kerren (Ed.), Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications — Vol. 3, IVAPP, SciTePress, Portugal, 2020, pp. 149–156, http://dx.doi.org/10.5220/0008945201490156, URL http://www.ivapp.visigrapp.org/.

[22] A. Krause-Glau, M. Hansen, W. Hasselbring, Supplementary data for: Collaborative program comprehension via software visualization in extended reality, 2021, http://dx.doi.org/10.5281/zenodo.5790175.

[23] A. Krause, M. Hansen, W. Hasselbring, Live visualization of dynamic software cities with heat map overlays, in: Proceedings of the 9th IEEE Working Conference on Software Visualization, VISSOFT 2021, IEEE, 2021, pp. 125–129, http://dx.doi.org/10.1109/VISSOFT52517.2021.00024.

[24] A. Henrysson, M. Billinghurst, M. Ollila, Face to face collaborative AR on mobile phones, in: Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR'05, 2005, pp. 80–89, http://dx.doi.org/10.1109/ISMAR.2005.32.

[25] A. Schreiber, L. Nafeie, A. Baranowski, P. Seipel, M. Misiak, Visualization of software architectures in virtual reality and augmented reality, in: 2019 IEEE Aerospace Conference, 2019, pp. 1–12, http://dx.doi.org/10.1109/AERO.2019.8742198.

[26] F. Fittkau, A. Krause, W. Hasselbring, Software landscape and application visualization for system comprehension with ExplorViz, Inf. Softw. Technol. 87 (2017) 259–277, http://dx.doi.org/10.1016/j.infsof.2016.07.004.

[27] W. Hasselbring, A. Krause, C. Zirkelbach, ExplorViz: Research on software visualization, comprehension and collaboration, Softw. Impacts 6 (2020) http://dx.doi.org/10.1016/j.simpa.2020.100034.

[28] C. Zirkelbach, A. Krause, W. Hasselbring, Modularization of research software for collaborative open source development, in: Proceedings of the 9th International Conference on Advanced Collaborative Networks, Systems and Applications, COLLA 2019, 2019, pp. 1–7, URL https://www.thinkmind.org/index.php?view=article&articleid=colla_2019_1_10_50005.

[29] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, D. Kröger, Microservice decomposition via static and dynamic analysis of the monolith, in: Proceedings of the IEEE International Conference on Software Architecture Companion, ICSA-C, 2020, pp. 9–16, http://dx.doi.org/10.1109/ICSA-C50368.2020.00011.

[30] R. Wettel, M. Lanza, Visualizing software systems as cities, in: Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 92–99, http://dx.doi.org/10.1109/VISSOF.2007.4290706.

[31] C.L. Jeffery, The city metaphor in software visualization, in: Proceedings of the 27th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG 2019, 2019.

[32] V. Dashuber, M. Philippsen, Trace visualization within the software city metaphor: A controlled experiment on program comprehension, in: Proceedings of the 9th IEEE Working Conference on Software Visualization, VISSOFT 2021, 2021, pp. 55–64, http://dx.doi.org/10.1109/VISSOFT52517.2021.00015.

[33] F. Fittkau, S. Finke, W. Hasselbring, J. Waller, Comparing trace visualizations for program comprehension through controlled experiments, in: Proceedings of the 23rd IEEE International Conference on Program Comprehension, ICPC 2015, 2015, pp. 266–276, http://dx.doi.org/10.1109/ICPC.2015.37.

[34] F. Fittkau, J. Waller, C. Wulf, W. Hasselbring, Live trace visualization for comprehending large software landscapes: The ExplorViz approach, in: 1st IEEE International Working Conference on Software Visualization, VISSOFT 2013, 2013, pp. 1–4, http://dx.doi.org/10.1109/VISSOFT.2013.6650536.

[35] F. Fittkau, E. Koppenhagen, W. Hasselbring, Research perspective on supporting software engineering via physical 3D models, in: Proceedings of the 3rd IEEE Working Conference on Software Visualization, VISSOFT 2015, IEEE, 2015, pp. 125–129, http://dx.doi.org/10.1109/VISSOFT.2015.7332422.

[36] F. Fittkau, E. Koppenhagen, W. Hasselbring, Experimental data for: Research perspective on supporting software engineering via physical 3D models, 2015, http://dx.doi.org/10.5281/zenodo.18378.

[37] M. Bandukda, Z. Nasir, Efficacy of distributed pair programming, in: Proceedings of the 2nd International Conference on Information and Emerging Technologies, ICIET 2010, 2010, pp. 1–6, http://dx.doi.org/10.1109/ICIET.2010.5625667.

[38] K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, second ed., Addison-Wesley Professional, Boston, 2004.

[39] J. Brück, Collaborative program comprehension based on virtual reality, 2020, http://dx.doi.org/10.5281/zenodo.3923715, Kiel University.

[40] J. Brück, C. Zirkelbach, Thesis artifacts for: Collaborative program comprehension based on virtual reality, 2020, http://dx.doi.org/10.5281/zenodo.3923715, Zenodo.

[41] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J.M. González-Barahona, M. Lanza, Codecity: On-screen or in virtual reality? in: Proceedings of the 9th IEEE Working Conference on Software Visualization, VISSOFT 2021, 2021, pp. 12–22, http://dx.doi.org/10.1109/VISSOFT52517.2021.00011.

[42] R. Souza, B. da Silva, T. Mendes, M. Mendonça, SkyscrapAR: An augmented reality visualization for software evolution, in: Proceedings of the 2nd Brazilian Workshop on Software Visualization, WBVS 2012, 2012.

[43] M. Hansen, Collaborative program comprehension based on augmented reality, 2021, Kiel University, URL http://oceanrep.geomar.de/53848/.

[44] M. Hansen, Evaluation results — Colaborative Program Comprehension based on Augmented Reality (Master's Thesis), Zenodo, 2021, http://dx.doi.org/10.5281/zenodo.5075126.

[45] L. Merino, The Medium of Visualization for Software Comprehension (Ph.D. thesis), University of Bern, 2018, http://dx.doi.org/10.7892/boris.118274.

[46] L. Merino, M. Hess, A. Bergel, O. Nierstrasz, D. Weiskopf, PerfVis: Pervasive visualization in immersive augmented reality for performance awareness, in: Companion of the 10th ACM/SPEC International Conference on Performance Engineering, ICPE 2019, 2019, pp. 13–16, http://dx.doi.org/10.1145/3302541.3313104.

[47] A. Elliott, B. Peiris, C. Parnin, Virtual reality in software engineering: Affordances, applications, and challenges, in: Proceedings of the 37th IEEE/ACM IEEE International Conference on Software Engineering, Vol. 2, ICSE 2015, 2015, pp. 547–550, http://dx.doi.org/10.1109/ICSE.2015.191.