Collaborative Program Comprehension based on Virtual Reality

Bachelor's Thesis

Johannes Brück

April 30, 2020

Kiel University
Department of Computer Science
Software Engineering Group

Advised by: Prof. Dr. Wilhelm Hasselbring
Christian Zirkelbach, M.Sc.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 30. April 2020

Abstract

To meet the increasing requirements of modern computer programs, the complexity of software systems is constantly growing. Monitoring tools, especially tools for software visualization, help to handle this complexity and to gain a deep understanding of the structure and the runtime behavior of software systems. The VR extension of the research tool ExplorViz provides an approach to collaboratively exploring large software landscapes using virtual reality. A first practical test has shown that the tool meets the basic usability requirements.

The aim of this thesis is to investigate whether the VR extension is suitable for collaboratively solving complex tasks in the context of program comprehension. For this purpose we extended the VR extension with a number of features that increase the information offer and enhance the possibilities of interaction. Subsequently, we conducted an experiment with 24 participants, in which software analysis tasks were to be solved in teams. The results show that the VR extension is well suited for collaboratively analyzing both static and dynamic aspects of software systems. However, the experiment also revealed problems in navigation and layout.

Contents

Chapter 1

Introduction

1.1 Motivation

The requirements for modern computer programs are constantly increasing. To meet these requirements, the complexity of the underlying software systems is growing. Monitoring and analysis tools help to handle this complexity. Especially software visualization tools can help to gain a deep understanding of structure and runtime behavior. Due to increasing computing power, new possibilities of immersive techniques have emerged. The research tool ExplorViz[1] [Fittkau et al. 2017] combines software visualization [Zirkelbach et al. 2019b] with immersive experience by allowing developers to explore entire software landscapes in virtual reality (VR) [Zirkelbach et al. 2019c]. Since industrial software in particular is usually developed in teams, it is reasonable to analyze software systems collaboratively. According to this conclusion, the virtual use of ExplorViz was extended to multiple users [Hansen 2018]. A first practical test has shown that the multi-user extension of ExplorViz meets the basic requirements of usability [König 2018]. However, the question of whether ExplorViz is suitable for collaboratively solving complex problems of program understanding remains unresolved. This thesis aims at further investigating this question. In order to do this we will conduct an experiment with several participants, in which application-related tasks in the context of software analysis will be solved collaboratively. In preparation we will analyze the current state of the ExplorViz VR extension and extend it with features regarding usability and possibilities of interaction.

1.2 Goals

The main goal of this thesis is to evaluate wether the ExplorViz VR extension is suitable for collaborative program comprehension.

1.2.1 G1: Requirement Engineering

Our first goal is to analyze the current state of the ExplorViz VR extension and identify realizable features and modifications that might improve a collaborative use. On the one hand, our intention is to improve the usability of the extension. On the other hand, it is our

[1] https://www.explorviz.net/

aim to integrate more information on the analyzed software system into the visualization and to extend the interaction possibilities. We will select a subset of suitable features for the realization according to these criteria.

1.2.2 G2: Design

Based on the requirements analysis in G1, we will derive concrete adaptations from the selected features. Therefore, we will specify the features.

1.2.3 G3: Implementation

Our goal is to implement the features specified in G2.

1.2.4 G4: Evaluation

We will test the adapted version of the VR extension in an interactive experiment. For this purpose, the participants will be divided into teams of two to solve tasks with different use cases and difficulties in the context of software analysis. Based on the results, we will evaluate the features implemented in G3 as well as answer the question whether the VR extension is suitable for collaborative analysis of software systems.

1.3 Document Structure

In Chapter 2 we will explain the technical foundations that are relevant in the context of the thesis. Here we will primarily introduce the ExplorViz VR extension. Then we will discuss possible features and modifications in Chapter 3 before we specify a design in Chapter 4. Chapter 5 deals with the implementation of the features. Afterwards we explain the evaluation in Chapter 6 which constitutes an essential part of this thesis. Finally, we refer to related work in Chapter 7 and provide a summary of the thesis as well as an outlook for future work in Chapter 8.

Chapter 2

Foundations and Technologies

2.1 Virtual Reality

The term virtual reality describes computer generated simulations of environments in which the user can interact in three-dimensional situations [Coates 2019]. Head-mounted displays (HMD) enable the experience of those environments via head and position tracking. Thereby HMDs such as the Oculus Rift[1] provide a truly immersive experience [Desai et al. 2014]. There are various HMDs with differing advantages and disadvantages [Mehrfard et al. 2019]. However, a comparison by Mehrfard et al. [2019] has shown that the HTC Vive Pro[2] scores best in terms of comfort, display quality and compatibility with glasses. Even though numerous innovations and enhancements in connection with VR are discussed these days, the basic concept is not new. The first HMD was introduced by Ivan Sutherland in 1968 [Sutherland 1968]. Nonetheless, the opportunities of VR applications were raised on a higher level due to the increase of computing capacity.

In this thesis we will use VR as a setting to visually represent software systems and to explore software in a collaborative manner.

2.2 ExplorViz

ExplorViz is a tool for live monitoring and visualization of software landscapes based on the city metaphor [Fittkau et al. 2017]. The core functionalities are summarized in the following.

2.2.1 General Architecture

ExplorViz 1.5.0 uses monitoring data to visualize a software landscape and its components [Zirkelbach et al. 2019b]. Additionaly ExplorViz gives further information on program behavior. The data for that is provided by Kieker[3]. Kieker is a software that analyzes the runtime behavior of a system [van Hoorn et al. 2012]. Thereby Kieker detects control flows and corresponding temporal aspects (e.g., response time) [Rohr et al. 2008] which are the

[1] https://www.oculus.com/rift/
[2] https://www.vive.com/de/
[3] http://kieker-monitoring.net/

basis for the dynamic system visualization of ExplorViz. However, the tool for logging and extracting useful data on system behavior, i.e., Kieker, can simply be substituted since the system structure of ExplorViz was updated to a Microservice Architecture during development [Zirkelbach et al. 2018]. Thus, ExplorViz has a more modular composition, and it is thereby partly exchangeable and easy to extend [Zirkelbach et al. 2019c]. Figure 2.1 shows the architecture of ExplorViz.

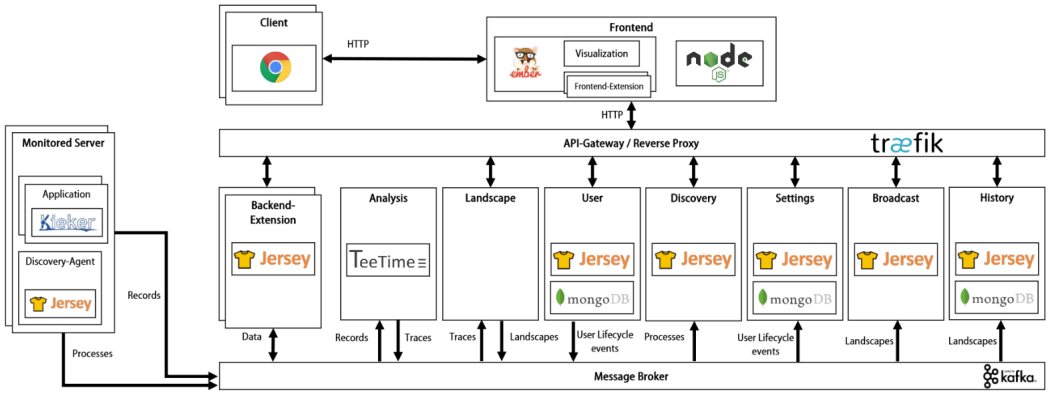


Figure 2.1. ExplorViz 1.5.0 software stack

The backend consists of several microservices which send and receive data via the message broker. The data exchange between the backend services and the frontend is performed via an API gateway. The frontend employs the web framework Ember.js[4]. The source code of the frontend component consists of JavaScript[5] code for the most part. The 3D graphics are animated using the JavaScript library Three.js[6].

2.2.2 Funcitonalities and Perspectives

ExplorViz offers two different perspectives on the analyzed software: the landscape perspective focuses on system comprehension, the application level perspective focuses on program comprehension [Fittkau et al. 2017].

On the landscape level the individual systems and the communication between them are visualized (see Figure 2.2). The interior of the grey colored systems contains node groups or nodes which are represented by the color green. Again, nodes may contain one or more applications. They are indicated by the color blue. Orange lines between those applications show the communication.

[4]https://emberjs.com/
[5]https://www.javascript.com/
[6]https://threejs.org/

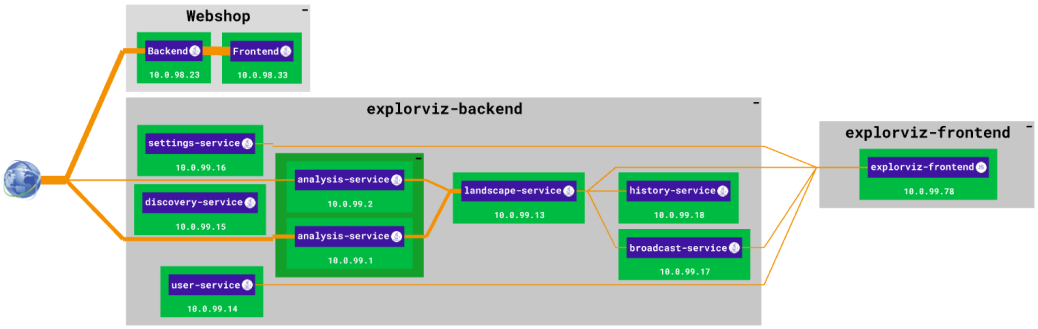


Figure 2.2. Landscape perspective of ExplorViz 1.5.0

By clicking one of the blue application boxes the user can easily change to the more detailed application level visualization (see Figure 2.3). In that, the packages and their contained elements are shown. The lowest element of this hierarchy is a class, which is colored in purple. The individual forming and size of a class gives information on the number of active instances. Likewise, the thickness of the communication links represents the count of requests and messages between two applications. Based on this, ExplorViz enables the monitoring of whole traces within the system. By viewing infoboxes it is possible for the user to learn further structural or dynamic data of the individual system entities. For applications for example the time of the last use and the programming language is displayed. For nodes, information about the RAM usage and the utilization of the CPU is displayed. In the application perspective for example the counted operation calls and the average response time of those are provided on request.

ExplorViz provides other services to get a deeper understanding of the system performance, e.g., in connection with database queries [Fittkau et al. 2017]. However, they are not discussed here, because the thesis concentrates on the topics above.

2.2.3 VR Extension

It is also possible to visualize both the landscape and the application level perspective using VR [Zirkelbach et al. 2019a]. Thereby the user can immersively explore the analyzed software somehow in a real space, where the visualized landscape is placed on the floor [Häsemeyer 2017]. The user interaction has been implemented by the use of gesture control in a first approach [Fittkau et al. 2015] and later also by using the controllers of the individual virtual reality set. In the thesis we will take a look at the interaction using controllers.

The VR extension for example provides a teleportation function inside the virtual room. Additionally, the VR extension enables the user to open individual applications of the

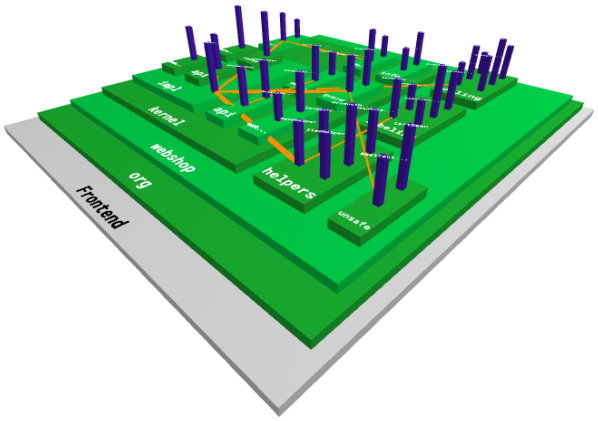


Figure 2.3. Application perspective of ExplorViz 1.5.0

different systems as a moveable flying object. The components of the application can be shown either closed or in detail with the containing components and classes inside them, as in the conventional web frontend. Furthermore, the user can select one component at a time to highlight it in color. On top of the VR extension in version 1.1 another update was developed which includes the possibility to explore the software model collaboratively as a main modification [König 2018]. Thus, multiple users can enter the virtual room and both examine the model and operate on it. The participants of a session are able to see each other in the way that a virtual image of their HMD including the moving controllers appear (see Figure 2.4).

In addition to the user visualization other settings were developed. A user can change via a personal menu the camera perspective, move the landscape, spectate other participants or set the connection properties. Furthermore, different text messages are displayed to inform the user about the success of the own actions and the connection status. In multi-user mode, the selection function also serves as a means of interaction with other users. Each user has an indiviual color.

The VR extension is technically realized by extending the project with a backend extension on one hand and a frontend extension on the other hand [Hansen 2018]. The frontend is extended by an add-on in the ember structure (see Figure 2.1) which integrates seamlessly into ExplorViz the corresponding models and rendering for the VR environment. The backend extension is realized by an additional microservice in the software stack (see Figure 2.1). The backend stores the necessary user and landscape data and provides an API for data exchange and model updates between users.

ExplorViz is a browser-based application, the VR extension uses the JavaScript API

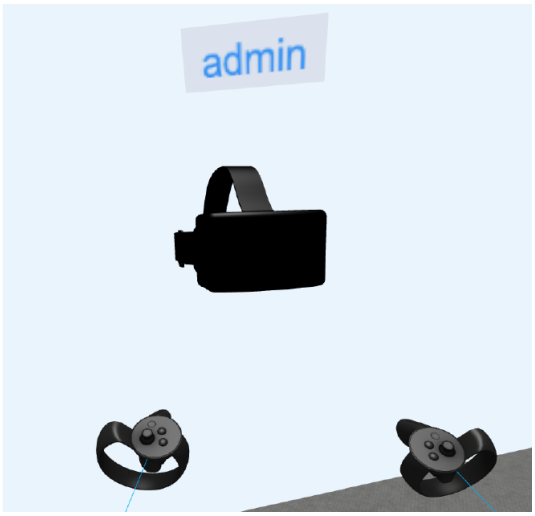


Figure 2.4. Visualization of other user including HMD, controllers and & colored name [Hansen 2018]

WebVR[7] to enable interaction with the VR devices. Until now the VR extension supports the Oculus Rift and the HTC Vive (Pro).

The VR extension of ExplorViz is the subject of this thesis. We will add features to improve usability based on those extensions. Finally, we will use ExplorViz to conduct our later experiment to evaluate collaborative software analysis using VR.

[7] https://webvr.info/

Chapter 3

Requirement Engineering

Our overall goal is to evaluate collaborative exploration of software and program comprehension using VR in ExplorViz. As a basis for a meaningful study we need to create the highest possible usability, i.e., an easy understanding of the functions and an intuitive interaction with the virtual environment. In addition, we aim to extend the possibilities to obtain information about the visualized system landscape and to operate on it in order to perform more complex tasks in the later experiment, which require the cooperation of the participants.

In the following we will elaborate suitable features. Then we will choose a subset of those features considering relevance and feasibility.

3.1 Results of Previous Studies

In order to identify potential features that are not yet implemented or in need of improvement, we will consult past work related to the VR extension of ExplorViz. A first orientation is provided by the work of Hansen [2018] and König [2018] on the multi-user extension of the VR extension. In order to test the multi-user extension, Hansen and König have conducted an experiment similar to the one we are planning, but with a focus on evaluating the usability of their work. The following potential features result after analysis of the development process and the experiment. The potential features are numbered:

PF1: Make the movement of the landsape more intuitive, e.g., by using the trackpad of the VR controller.

PF2: Enable single-handed usage of the options menu.

PF3: Include some kind of instruction manual to the handling of the environment and the assignment of keys.

PF4: Enable selection of opened components.

PF5: Display other participants of a session as a whole, i.e., display avatars or abstractions of them.

PF6: Improve readability of labels and messages, e.g., by changing the font size.

PF7: Solve the problem of blank screen or freeze triggered by (re)rendering of the landscape.

3.2 Comparison with the Desktop Visualization

The VR extension is an extended service of the ExplorViz software, which has been developed step by step. Therefore, the conservative desktop visualization offers features that the VR extension does not implement. The following potential features result from the comparison between the VR visualization and the desktop visualization:

PF8: Include a tutorial that helps in understanding the handling and the features of ExplorViz.

PF9: Add a view to get further information on communication lines, e.g., number of requests and the average response time.

PF10: Adapt the infoboxes of the entities to the format of the desktop visualization (including mathematical units).

PF11: Add an option to examine traces.

PF12: Enable live visualization of the landscape (including pauses). Enable selection of a specific point in time within the VR environment.

3.3 Selection of Features

Moving the landscape is not essential in order to interact with it, as the user can change his personal view of the landscape by teleportation or movement in physical space. For this reason we assign a lower priority to Feature PF1. Since the use of the menu is mainly for initial settings, we draw the same consequence for Feature PF2. We consider the possibility to display an overview of the individual controller assignment within the VR environment (Feature PF3) to be very important, especially for a self-explanatory use of the VR extension as well as a smooth running of the experiment. Since feature Feature PF4 is expected to be easy to implement and would increase the usefulness of the selection function, we will also include it. Displaying other users as avatars (Feature PF5) will again have a lower priority. So far, the participants of a session are only displayed in the form of the virtual device, i.e., controllers and headset. Even though this could well lead to a more realistic visualization of the situation, a realistic representation of the movements seems primarily relevant to us for an immersive experience. Feature PF6 does not require a generalized solution, since the readability of individual components depends primarily on the quality and resolution of the respective VR display. After analyzing the reloading problem (Feature PF7) we consider the problem as being complex and non-trivial and unsolveable by simple multithreading.

A tutorial to get familiar with the VR extension (Feature PF8) could be helpful for general use. However, for our experiment it might also be advantageous to provide a non-implemented introduction to VR extension to prepare the users for their tasks. For this reason and due to its complexity we assign less importance to the feature. More detailed data on communication lines (Feature PF9) would increase the information offer and the meaningfulness of the software model, which is one of our goals. From the holistic perspective on ExplorViz, the adaptation of the infoboxes to those of the desktop visualization (Feature PF10) is also relevant. This way a uniform syntax and a good comparability between the desktop visualization and the VR visualization is established. The display of traces (Feature PF11) as well as the live visualization (Feature PF12) would be the next step to adapt the VR extension to the conservative visualization and to exploit the scope of the monitoring data, but it would go beyond the complexity of the planned study.

In addition to the features mentioned above, there are other functions that we consider useful. First of all, it would be consistent to treat communication lines in the same way as other entities of the landscape. This means that besides calling infoboxes, class communication lines should also be selectable, i.e., markable in color. In addition, entities are locally highlighted, i.e., temporarily darkened at the moment when the ray of a controller points at them. It would improve the usability if entities were highlighted according to the provided controller functions. Apart from that, we find it helpul to integrate a general reset function that allows us to reset the landscape and user position within the VR environment. From a developer's perspective, this would especially simplify the testing of the application. In summary, we selected the features presented in the following:

F1: Display of the key mapping

F2: Selection of opened components and class communication

F3: Uniform infoboxes and infoboxes for communication

F4: General reset option

F5: Highlighting according to controller functions

Chapter 4

Design

In this chapter we will specify the design and planned implementation of the previously selected features.

4.1 F1: Display of the Key Mapping

To support (first time) users of the VR extension, we will display the key mapping. This is especially useful for the experiment because the user does not have to interrupt the VR experience to learn about the controls. Since the VR extension is implemented for both the HTC Vive and the Oculus Rift, we will integrate the respective key bindings depending on the hardware employed by a user when connecting. We will also distinguish whether the user is in standard right-handed mode or has selected left-handed mode in the settings.

The option menu has a sub-menu *Advanced Menu* in which the left-handed mode can be set. We will include another sub-menu called *Controls* which can be accessed in the *Advanced Menu* (see Figure 4.1) and then displays the corresponding key assignment as an image (see Figure 4.6 and Figure 4.8). Like the options menu, it should be linked to the left controller so that the user can zoom in on the image by moving the controller.

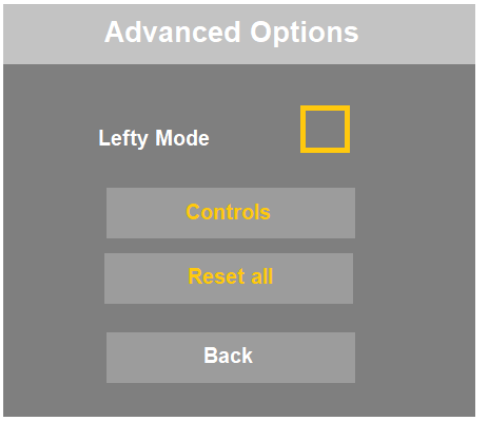


Figure 4.1. Visual concept for the extended *Advanced Menu*

In the following we explain how the labels in the key mapping (see Figure 4.6 and Figure 4.8) should be interpreted. Clicking the *Options Menu* button will open/close the options menu or, if a nested menu is open, return to the parent menu. Pressing the *Select/Teleport* trigger will select an object if the ray of the associated controller points at it. If the ray points at the floor, the user will be teleported to that position. Holding the *List all users* key will display all users in the current session, including their color and connection status. By activating the *Open/Close* trigger a component is opened/closed, if the ray intersects it. By holding the *Move application* key a 3D application object will be rotated according to the controller movement, if the ray intersects an entity of this application object. Clicking the *Display Information* key opens the infobox for an entity if the ray intersects it. If an infobox is already open, it will be closed.

4.2 F2: Selection of Opened Components and Class Communication

When selecting opened components only the component itself, i.e., the visible edge of the component should be colored. In particular, entities contained therein, that is, components or classes, should not be colored automatically (see Figure 4.2). This would not correspond to the concept that only one object may be selected at a time, and in order to provide clarity.

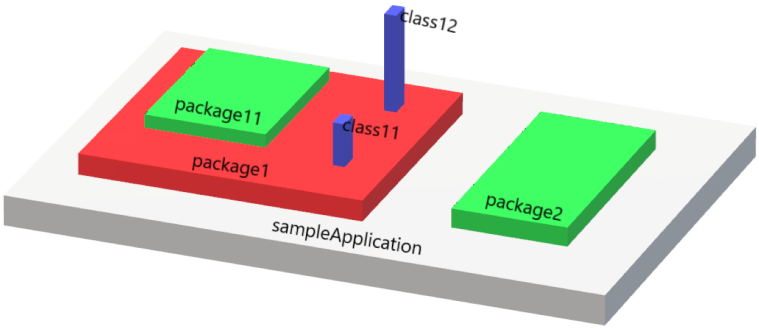


Figure 4.2. Visual concept for selection of opened components: *package1* is selected (red), the inner entities remain unchanged

In contrast to the conservative desktop visualization, we will not graphically highlight the affected classes when selecting the communication lines. The shape of a selected communication line also remains unchanged, only the color is altered (see Figure 4.3). In the application-level perspective ommunication always refers to classes. However, communication lines can be visible in the model even if the affected classes are not visible. This is the

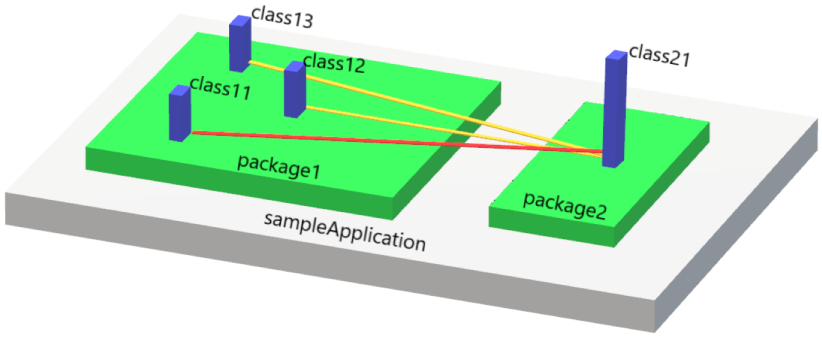


Figure 4.3. Visual concept for selection of class communication: Communication between *class11* and *class21* is selected (red)

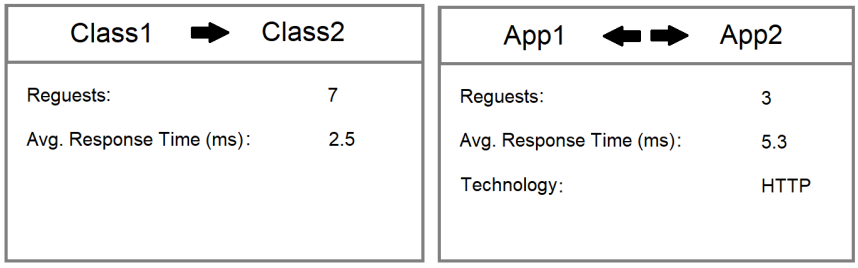
case if the classes are inside components that are closed. Communication lines should also be selectable in this case, since the analysis of communication can also be relevant if the application is examined at a high level of abstraction. So far, only one entity can be selected at a time. By extending the entities allowed for selection, we are not going to override this rule. In particular, this means that if an opened component or a communication line is selected, no other entity object may be selected in the model.

4.3 F3: Uniform Infoboxes and Infoboxes for Communication

In order to compensate the differences between the infoboxes of the desktop visualization and the VR visualization, node data must be adjusted first. This includes, on the one hand, the rounded percentage of CPU usage and, on the other hand, the total and free RAM converted into gigabytes. For class communication, the number of requests and the average response time in milliseconds should be displayed. For application communication, the technology used should also be shown. Furthermore, for application and class communication, the applications or classes concerned shall be displayed by name in the infoboxes, and the direction of communication shall be indicated by an arrow (see Figure 4.4). In the case of bidirectional communication, two arrows should be displayed in opposite directions.

4.4 F4: General Reset Option

The general reset function is intended to set the entire VR environment to the initial state. The initial state refers to the user position, i.e., the camera perspective, and the state of



(a) Class communication: *Class1* calls *Class2* with 7 requests and an average response time of 2.5ms

(b) Application communication: *App1* communicates bidirectionally with *App2* via HTTP with 3 requests and an average response time of 5.3ms

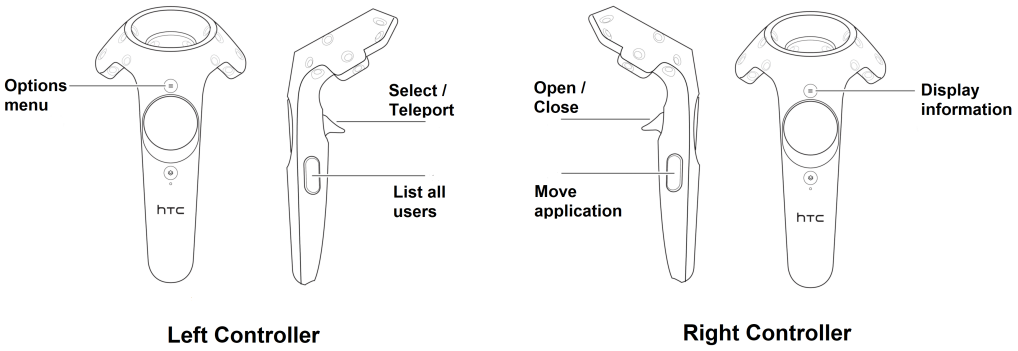
Figure 4.4. Visual concept for communication infoboxes

the software landscape. The camera is in its initial state when the height of the camera corresponds to the height of the HMD in physical space and the positioning is in the middle of the floor visualized in virtual space. The software landscape is in its initial state when all applications are closed and the all systems are closed and positioned as part of the landscape slightly away from the user. However, this does not mean to re-initiate the entire rendering procedure, as this would result in a re-connection to the hardware and lead to an explicit entering of the virtual space by computer interaction. Instead, the idea is that we reset the landscape and user position and close all systems and applications explicitly. It should be noted that closing an application automatically includes closing all inner components as well as deselecting all contained entities. However, we do not provide this option in multi-user mode, as it might be problematic if one user would perform the operation with consequences for all others. Furthermore, the question would remain whether a user should influence the position of the other users, i.e., their camera perspective, or if this would lead to confusion.

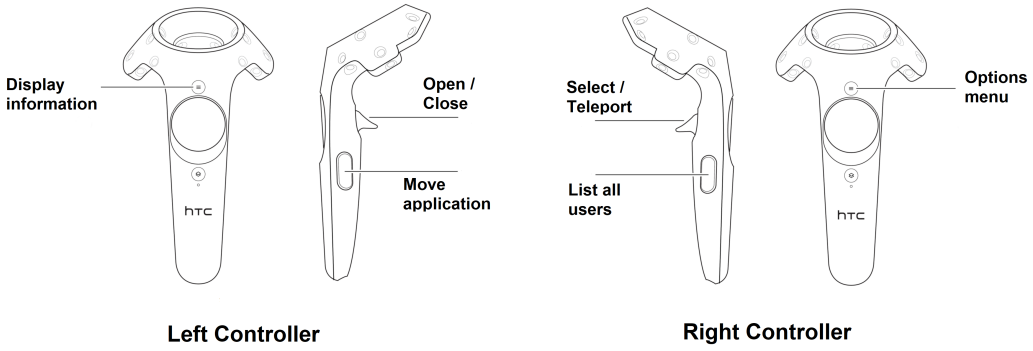
The reset option should also be available in the *Advanced Menu* (see Figure 4.1).

4.5 F5: Highlighting according to Controller Functions

For intuitive use, the highlighting should take place when the ray of a controller points at an entity, which provides a central function. Central functions are those initiated by activating the trigger of the respective controller. Accordingly, the central function for the primary controller (in standard right-handed mode the right-handed controller) is the opening/closing of components. The central function for the secondary controller (in right-handed mode, the left controller) is therefore the selection function (see Figure 4.6 and Figure 4.8).

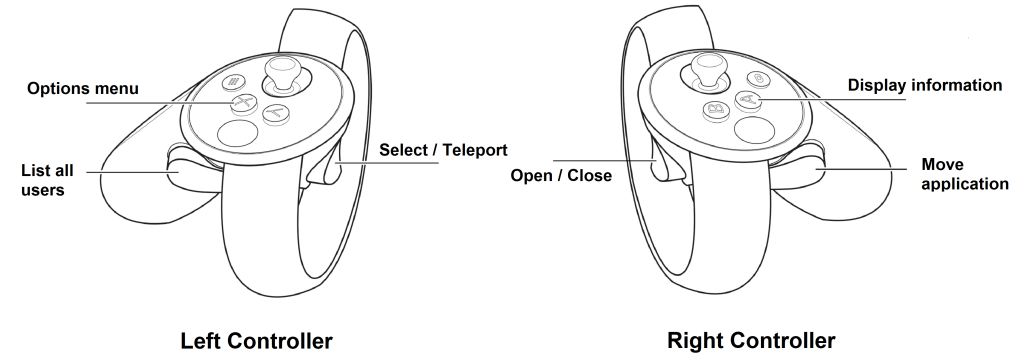


(a) Right-handed

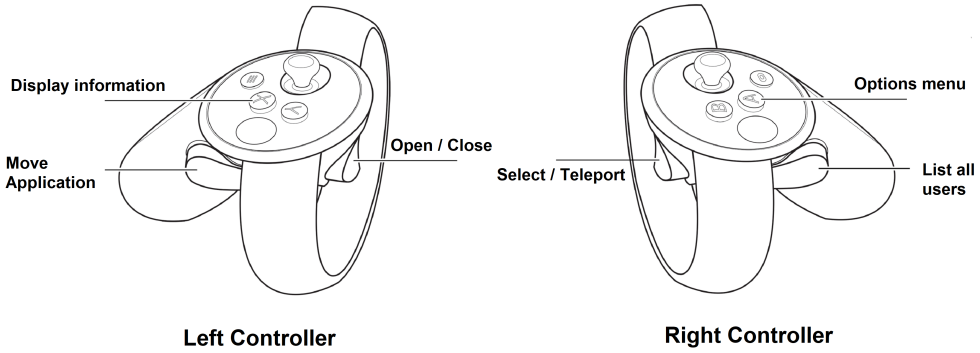


(b) Left-handed

Figure 4.6. HTC Vive (Pro) controls



(a) Right-handed



(b) Left-handed

Figure 4.8. Oculus Rift controls

Chapter 5

Implementation

The next step is to implement the features in preparation for the planned experiment. During the technical revision of the VR extension our goal is to create optimal conditions to enable a high individual usability and at the same time a high collaborative applicability. First, we will describe the concrete realization of the specified features from the previous chapter. Since the features mainly refer to changes in the user-specific presentation, the following adjustments will, unless otherwise specified, technically only affect the frontend extension. Afterwards, we will explain changes in the code that go beyond the planned features. These are due to technical problems that occurred during the development process.

5.1 Feature Implementation

5.1.1 F1: Display of the Key Mapping

To display the key assignment, we create another class *Controls-Menu*. The created class inherits from the same super class *Base-Menu* as all other menu classes to ensure a uniform design and logic. We identify the image to be loaded in a case distinction using the ID of the connected controller set and an internal flag, which tells us whether left or right-handed mode is set. We get both information from the existing service *User* in the frontend extension. When using headsets other than the HTC Vive (Pro) or the Oculus Rift, this procedure can be easily extended.

The canvas for the menus are created in the *Menu* class. For each instance, menu elements, e.g., buttons, are queued using the corresponding methods. The canvas is initially built, i.e., the menu elements from the queue are added when the respective menu is opened and updated whenever the ray of a controller intersects the menu, for example to enable hover effects. To add the image of the respective key mapping to the canvas, we create a new *addImage(...)* method (see Listing 5.1) in the *Menu* class, by which the image is packed into the queue (line 6). This way we extend the interface of the class for other needs as well.

Listing 5.1. Class *Menu*: Load image and update canvas

```
1 addImage(url, dx, dy, dWidth, dHeight) {
2     ...
3     image.onload = () => {
4       this.update();
5     };
6     this.get('items').push({ type: 'image', image, dx, dy, dWidth, dHeight});
7   }
```

Loading the image may be time consuming and we want to avoid that the image first appears on the canvas when the ray randomly intersects it, we update the canvas once the image is loaded (line 3). Figure 5.1 shows the menu after the image has been successfully loaded.

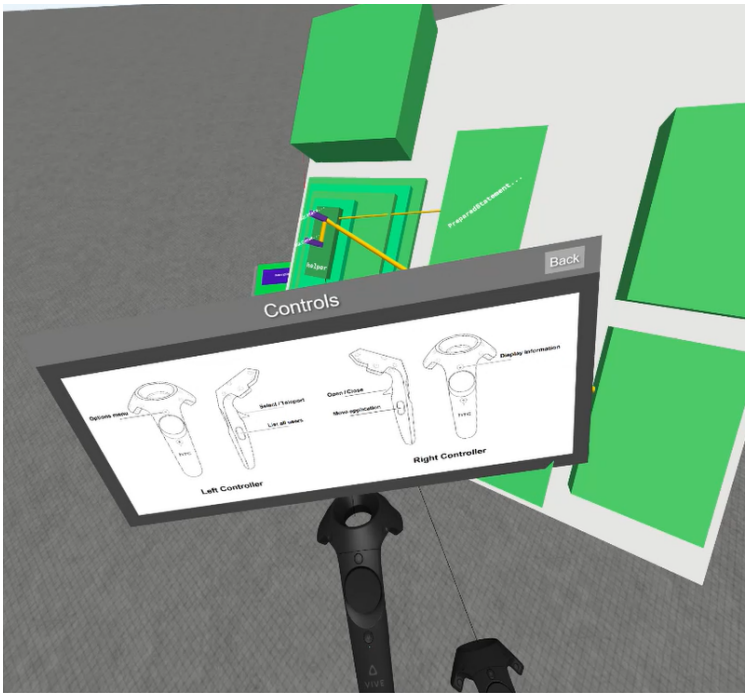


Figure 5.1. Controls menu for the HTC Vive

5.1.2 F2: Selection of Opened Components and Class Communication

The task of enabling the selection of opened components is trivial. Technically speaking, selecting means changing the color and remembering the original color and the selected object

for later deselection (this can be triggered by explicitly selecting the same object or implicitly by selecting another object). Therefore, it makes no difference for the implementation whether the component is marked as open or closed in the internal model.

Selecting the class communication lines is more complicated. Indeed, we can identify the communication line when clicking the select function and pointing the corresponding ray to the graphical object and thus color it (see Figure 5.2). However, during the development we discovered that we cannot query the ID of the communication model. Therefore, if we need to implicitly deselect the object, it is not easy to identify it. We suspect that the ID is either not directly linked to the graphic object or is lost during the rendering process.

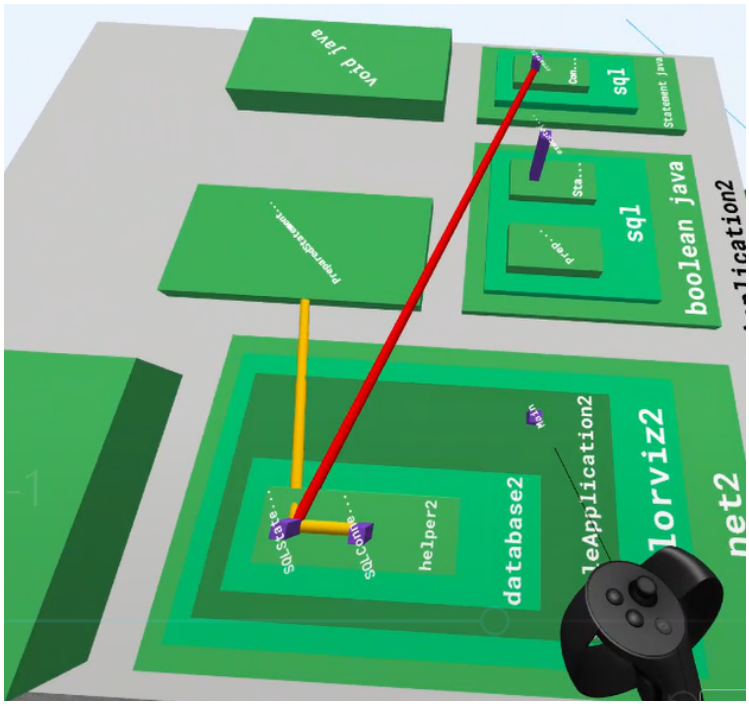


Figure 5.2. Selection of class communication (red)

As a solution, we identify communication lines for this purpose not via an ID, but via their source and target class, which can be queried as attributes in the underlying model. The reason that this is an actual identifier is due to the fact that class communication is always aggregated internally in relation to a source and a target class, regardless of whether the communication is bidirectional or not (this information is stored in a flag and visualized via arrows).

5.1.3 F3: Uniform Infoboxes and Infoboxes for Communication

For the adapted representation of the already existing infoboxes, mathematical rounding and conversion of units suffice. The availability of infoboxes for communication lines can be enabled analogous to those of other entities. The necessary data is queried in the model of the core frontend and then packed into a graphic object. During the development we noticed that the design of the infoboxes differs strongly from the design of the options menu (see Figure 4.1) or the message boxes. Therefore, and motivated by the idea to reuse as many code components as possible, the visible infoboxes are now running instances of a new sub class of the general *Base-Menu* class. Figure 5.3 shows the infobox for class communication in the new design.

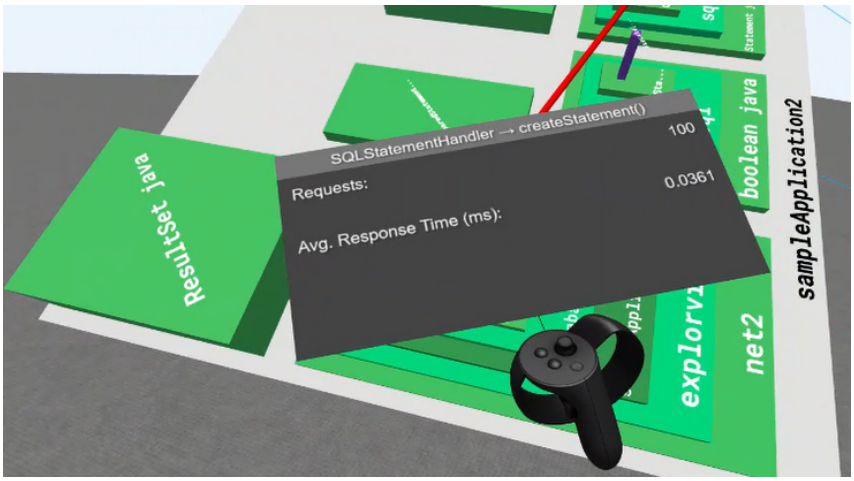


Figure 5.3. Infobox for class communication

5.1.4 F4: General Reset Option

For the implementation of the general reset option we integrate a listener for events, triggered in the *World* class, which offers services concerning the landscape, into the main rendering class *VR-Rendering*. If the user now clicks the reset button (see Figure 4.1), a method is called in the *World* service, which then triggers a corresponding procedure in the mentioned listener. This approach allows all classes of the frontend extension to perform a general reset, if necessary. Listing 5.2 shows what happens in this case.

Listing 5.2. Class *VR-Rendering*: Listener for general reset

```
1     this.get('world').on('resetAll', () => {
2       this.removeOpenApps();
3       this.get('world').resetLandscape();
4       this.resetLanscape();
5       this.get('localUser').resetPosition();
6       return;
7     })
```

In line 2 all open applications are closed. Afterwards the position of the system landscape is reset (line 3) and all open systems are closed (line 4). Finally, the position of the user and thus implicitly the perspective of the camera is reset (line 5).

5.1.5 F5: Highlighting according to Controller Functions

In order to adapt the highlighting of the entities to the controller functions, we have to check the type of the entity and other relevant conditions as soon as the ray of a controller intersects an entity. For the primary controller (see Listing 5.3) we highlight the entity only if it is a component (line 3) since only these can be opened or closed. Since the foundation of a 3D application model is also stored internally as a component, we have to exclude it (line 4). Furthermore, highlighting is prevented if the application is currently bound to the controller for rotation (line 5).

Listing 5.3. Class *Interaction*: Highlighting according to primary controller function

```
1   checkIntersectionPrimaryController(objects) {
2       ...
3       if ((emberModelName === "component"
4        && !emberModel.get("foundation"))
5         && !this.get('app3DBinded')) {
6       // highlight entity
7       }
8 }
```

For the secondary controller (see Listing 5.4) we highlight an entity if it is a component (line 3), class (line 4) or class communication (line 5) because the selection function is available for these only. Analogous to the condition for the primary controller, we check whether the entity is a foundation (line 3) or the application is bound to a controller (line 6).

Listing 5.4. Class *Interaction*: Highlighting according to secondary controller function

```
checkIntersectionSecondaryController(objects) {
    ...
    if (((emberModelName === "component" && !emberModel.get("foundation"))
     || emberModelName === "clazz"
      || emberModelName === "drawableclazzcommunication")
       && !this.get('app3DBinded')) {
    // highlight entity
    }
}
```

5.2 Unexpected Issues

5.2.1 Propagation of Camera Position and Rotation

While testing the VR Extension with two users we noticed that the visualization controllers of the other user moved and rotated in virtual space as we expected. But this was not true for the headset. Since this was a successfully implemented feature of previous works, this problem is probably due to an update of THREE.js or the WebVR API.

The procedure for position updates in multi-user mode works in the following way. The local camera object, which also indicates the position of the headset in virtual space, is stored in the local *User* service. The camera position is continuously queried and compared to the last memorized position. If the position changes, the new data is sent to the backend extension and from there is propagated to the other clients. Each client instantiates locally all other users as *VR-User* objects. Each *VR-User* holds the position data of the respective user which are overwritten by requests received from the backend extension. The position data is the basis for the graphical visualization of the respective HMD. The procedure is analogous for the rotation .

However, this leads to the problem that the global position of the local camera object in virtual space seems to be independent of the headset tracking when queried. Instead, the data is only updated when explicit position updates are performed in the frontend extension, e.g., when teleporting or changing the camera height in the options menu.

Listing 5.5. Class *Multi-User*: Position and rotation computation

```
let matrix = this.get('localUser.camera.matrixWorld').clone();

const posCamera = new THREE.Vector3();
this.get('localUser.camera').getWorldPosition(posCamera);
```

```
5  let posCameraMatrix = new THREE.Vector3(matrix.elements[12],matrix.elements[13],
       matrix.elements[14]);
6  posCamera.add(posCameraMatrix);
7
8  let cameraQuaternion = this.getQuaternionFromMatrix(matrix);
9
10 // Send data to Backend Extension
```

In order to receive the correct data (see Listing 5.5), the position data of the camera object in virtual space (line 4) must be combined, i.e., added (line 6), with the tracked position data of the headset in physical space (line 5) which is stored in an internal matrix object (line 1). Since the rotation is not explicitly changed (e.g., by teleporting), it is sufficient to send the stored rotation based on the tracked headset instead of the rotation data of the camera object. For propagation, we use an alternative mathematical structure, the quaternion, in order to simplify the rotation updates of the graphical objects later on (line 8).

5.2.2 Propagation of Selection and Deselection

Another issue that occurred during testing is that selecting activities in multi-user mode is also visible to the other clients, while deselecting activities cause problems. If an object is selected in multi-user mode, the necessary data is sent to the backend extension in the form of a JSON[1] object and propagated from there to all clients (see Listing 5.6). For each client, the selection is then made for the given entity (line 5). The explicit deselection is performed equally. A flag indicates the type of operation to be performed (line 3).

Listing 5.6. Propagation of component or class selection

```
1 {
2  "userID": 1,
3  "isHighlighted": "false",
4  "appID": "landscape-38293",
5  "entityID": "landscape-93874",
6  "color": "rgb(255,0,0)"
7 }
```

For complete execution we have to make sure that the implicit deselection, i.e., if another object is selected, is performed additionally on the respective client.

Furthermore, if the object is a communication line, we must also send the corresponding source and target class for identification (see Listing 5.7, lines 6-7). We then overwrite the *entityID* to know that we need to look at the class IDs (line 5).

[1] https://json.org

Listing 5.7. Propagation of communication selection

```
{
 "userID": 1,
 "isHighlighted": "false",
 "appID": "landscape-38293",
 "entityID": "clazzcommunication",
 "sourceClazzID": "landscape-72443",
 "targetClazzID": "landscape-28593",
 "color": "rgb(255,0,0)"
}
```

Chapter 6

Evaluation

6.1 Goals

Based on the previous experiment by König [2018] and Hansen [2018], it can be stated that the VR extension of ExplorViz meets the basic usability requirements and is suitable for working in teams. The VR extension has been enhanced with some features and the aim of this work is to investigate the appicability in terms of collaboratively solving complex tasks. Our concrete research questions are the following:

- ▻ Are the newly added features a gain for the use of the VR extension?
- ▻ Is the VR extension suitable for solving complex tasks in the context of static and dynamic software analysis?
- ▻ Is the VR extension suitable for solving complex tasks in collaboratively in teams?

Based on the evaluation we verify our primary research question whether the VR extension is suitable for collaborative program comprehension.

6.2 Methodology

In order to achieve our goal, we are conducting an experiment with a sufficient number of test persons. In doing so, we do not perform a controlled experiment but create relevant scenarios and realistic problem cases in the context of system and software analysis and evaluate the success of the subjects according to appropriate measures. The above-mentioned experiment by König [2018] and Hansen [2018] serves as a model for the procedure, although we have other priorities. In analogy to their experiment, we organize the test persons into teams for solving assigned tasks, collect relevant data regarding individual prerequisites and ask for feedback.

6.3 Experiment

6.3.1 General Procedure

The procedure of the experiment for a team of two is divided into four phases: The preparation phase, the training phase, the assignment phase and the feedback phase. In the preparation phase we first inform each subject about the system to be analyzed. In the next step, each participant answers a questionnaire on his or her own that contains questions on personal data. After that each subject is informed about the ExplorViz tool. In the training phase we conduct a practical tutorial in order for the test persons to learn how to use the VR extension. In the following assignment phase the participants proceed to the solution of the respective tasks. Finally, in the feedback phase, each participant answers a questionnaire on his or her personal perception of the experiment.

6.3.2 Setup

For the duration of the experiment, the participants of each team are distributed in two different rooms, henceforth referred to as room A and room B. Figure 6.1 and Figure 6.2 show the experimental setup for the respective room.

Figure 6.1. Experimental setup for room A

Figure 6.2. Experimental setup for room B

For the VR visualization the training and assignment phase require one computer in each room which acts as a client, and a VR set. The specifications of the hardware used are shown in Table 6.1.

Table 6.1. Hardware specifications of the clients

	Room A	**Room B**
CPU	Intel Core i5-6500	Intel Core i7-6700HQ
RAM	16 GB	16 GB
GPU	NVIDIA GeForce GTX 1070	NVIDIA GeForce GTX 1060
OS	Windows 10 Pro	Windows 10 Enterprise
VR device	HTC Vive Pro	Oculus Rift
Tracking	2 base stations	2 Oculus Sensors

Furthermore, we use a separate server for the ExplorViz application with the configuration shown in Table 6.2.

The server runs the ExplorViz backend and frontend in version 1.5.0 and the frontend and backend VR extension in version 2.0. We also use the Colorpicker[1] extension in the

[1]https://github.com/ExplorViz/explorviz-frontend-extension-colorpicker

Table 6.2. Hardware specifications of the server

	Server
CPU	Intel Core i5-6500
RAM	16 GB
GPU	NVIDIA GeForce GTX 1070
OS	Windows 10 Pro

version 1.5.0 to adjust the colors of the visualization in case one of the participants is affected by a visual impairment. The clients use Firefex[2] version 72.0.2 since newer versions do not support the WebVR standard.

For the individual phases of the experiment, we use a comprehensive digital questionnaire [Brück 2020] that we have previously created using SurveyJS[3]. Each team and each individual participant is mapped with an ID in order to bring together the data and results of the individual phases afterwards. We conduct the expriment with two leaders. The answers of the respective teams to the tasks in the assignment phase are communicated orally and are filled in by the experiment leaders at the corresponding stage of the questionnaire as the test persons are impeded by the use of their HMDs. Therefore, but also for guidance and the tutorial, one of the experiment leaders is positioned in room A and one in room B. We enable communication via Discord[4] on the same channel. Oral communication is necessary for the participants to communicate within the experiment while solving tasks. Additionally, we use it to communicate the tasks and to note the answers. Furthermore, we recorded what each participant was seeing within his HMD, in order to see if there are correlations between their interaction and given answers.

For the tutorial we use the slightly modified sampleApplcation[5] [Brück 2020], which features to basically identical applications within, one for each user. In the assignment phase, a time section of the system CoCoMe[6] serves as landscape [Brück 2020].

6.3.3 Preparation Phase

Introduction to CoCoMe

At the beginning of the preparation phase we inform the test persons about the analyzed system CoCoMe. With the help of a video [Brück 2020], we explain the components of the system and possible use cases in general. In particular, we do not explain programming details such as package structure and classes of the application.

[2]https://www.mozilla.org/de/firefox/
[3]https://surveyjs.io/
[4]https://discordapp.com/
[5]https://github.com/ExplorViz/sampleApplication
[6]https://github.com/research-iobserve/cocome-cloud-jee-platform-migration

Personal Information

In the preparation phase we ask for personal information in a questionnaire. We survey the respective test person in three categories: Personal background, relationship to the team member and previous technical experience. The aim is to obtain relevant information about the participants, which we can later contextualize with the outcome of the assignment phase. We aim to identify correlations between the qualifications of the team and their success in solving problems. Based on this, we intend to create user profiles which are advantageous for working with the VR extension.

Regarding the personal background, we are not only interested in age and gender, but also in the professional and academic background. Specifically, we ask for the highest academic degree, the type of activity (i.e., study, research or industry) as well as its duration and affiliation.

In addition, we want to know the relationship between the team members since the tasks are later to be solved in collaboration. In doing so, we want to find out to what extent the team relationship has an influence on communication and thus indirectly on the success of the team. Therefore, we ask the following questions:

R1: How well do you know your team member?

R2: How often have you worked with your team member?

The possible answers to question R1 are *not at all*, *barely*, *well* and *very well*, to question R2 *never*, *sometimes* and *often*.

We are interested in the technical previous experience of the team members, which could be relevant for the understanding of the visualization, the use of the VR extension or generally for the processing of the tasks. Concretely we ask for experience in the following labeled fields:

E1: Java, C++, or similiar

E2: Software Architecture

E3: Software Development in Teams

E4: Static Software Analysis

E5: Dynamic Software Analysis

E6: Program Comprehension

E7: Reengineering and Reverse-Engineering

E8: Virtual Reality

E9: ExplorViz

E10: CoCoME

The test persons should classify their experience into the categories *None*, *Beginner*, *Intermediate*, *Advanced* and *Expert*, in order to differentiate exactly between the different levels of experience.

Finally, we ask about visual impairments as this could have a negative influence on the use of the VR extension and we can use this information to explain possible deviations in the results later on. If visual impairment occurs, we try to compensate with a suitable vision impairment color scheme

Introduction to ExplorViz

We complete the preparation phase by introducing the subjects to ExplorViz. This is mainly about the general visualization concept and especially not about how to interact with the visualization, especially in VR.. That means we inform the test persons about the purpose of ExplorViz, the different perspectives and the details of the visualization.

6.3.4 Training Phase

The idea of the tutorial is on the one hand to familiarize the participants comprehensively with the VR extension and to prepare them for the later processing of the tasks. This way we want to ensure that during the assignment phase the focus is on the actual processing of the tasks and not on experimenting with the features. On the other hand, the tutorial serves, apart from the individual previous knowledge, to create the same knowledge for all teams to make the results comparable. Furthermore, the tutorial is already conducted in multi-user mode, i.e., the team members are in the same virtual room and thus become additionally familiar with the multi-user features.

At the beginning of the tutorial the experiment leaders explain the general functionality of the HMDs, especially the tracking, and the basic movement in virtual space. Afterwards the experiment leaders explain again the visualization concept of the individual entities (i.e., systems, nodes, components, classes and communication) using the 3D model of the loaded sample application. In particular, they explain the dependence of the class height or communication thickness on the number of instances or requests. We also explain the following features step by step:

- ▻ Headset and controller visualization of other users
- ▻ Display of the key assignment (see Figure 4.6 and Figure 4.8)
- ▻ Teleportation
- ▻ Movement of applications
- ▻ Controller visualization, including laser pointers for targeting

- ▻ Highlighting according to controller functions
- ▻ Opening/Closing of components
- ▻ Selection of components, classes and communication lines, especially the user's own color
- ▻ Infoboxes for components, classes and communication lines, especially the arrow as an indicator for the request direction

6.3.5 Assignment Phase

In the assignment phase, the test persons are asked to solve a series of tasks together, using their acquired knowledge of VR extension and its use. Only one task at a time will be worked on in a given order. In each task slot, the experiment leaders read the task once at the beginning and, upon request, several times. In addition, they instruct the participants in case the complication-free processing is not possible, e.g., if the participants accidentally leave the tracked room or if the view is restricted, e.g., by accidently changing the camera height. Otherwise, the experiment leaders do not give any advice and especially do not repeat the visualization concept. On the one hand, we want to ensure complete comparability between the teams, on the other hand we want to keep communication between the teams and the experiment leaders to a minimum in order to simulate a realistic usage scenario. The test persons are allowed to communicate with each other without restrictions. A task is considered completed as soon as the team mentions an answer and explicitly titles it as the solution. Furthermore, the experiment leaders reserve the right to cancel a task slot with a warning if a certain time limit has been exceeded. In the following the concrete tasks are listed:

A1: Structure of the software system

- A1.1: Name all top-level packages within the application CoCoME. Top-level packages are packages on the highest hierarchy level.
- A1.2: Name all classes in the package *printer*. The package *printer* can be found under the package *org.cocome.cloud*.
- A1.3: Compare the packages *context* and *userdisplay*. Which of them contains more classes? The package *context* can be found under the package *org.cocome.tradingsystem*.

A2: System load inspection

- A2.1: Find the class with the highest number of instances within the package *org.cocome.cloud*. Provide the name of the class and the number of active instances.

A2.2: Find the the pair of classes with the highest number of requests between them. Both classes are located under the package *org.cocome.cloud.web*. Provide the name of both classes, the communication direction, and the number of requests.

A2.3: Find the communication between two classes with the highest average response time. Provide the name of both classes, the communication direction, and the average response time.

A3: Evolution of the software system

A3.1: Our survey system CoCoME must be evolved to support the supermarket chain's switch to RFID tagged products. Therefore the class *org.cocome.tradingsystem.cashdeskline.cashdesk.BarcodeScanner* will be replaced by a class for a RFID scanner and the CoCoME system must be altered to support this functionality. Describe the change within the application by naming all depended classes and their relation that may be affected by the change under the package *org.cocome.cloud.logic*.

The tasks cover different use cases and thus involve varying difficulties while solving them. Task A1 involves locating, naming and comparing packages and classes in terms of their structure. The solution of the tasks requires a basic understanding of the hierarchical visualization of packages and classes. Task A2 is about identifying bottlenecks in the system at runtime. Specifically we ask for classes with maximum instances and communication with maximum requests or response times. The solution of the tasks requires to understand the runtime behavior of the system. Task A3 deals with the evolution of the software system. In the case of a further development of the software by replacing a class and changing the underlying interface as a consequence, the classes affected by the changes have to be identified. The solution of the task requires that the team is able to identify and understand dependency relationships between classes based on the visualization of runtime behavior.

For each task we have defined appropriate solutions [Brück 2020].

6.3.6 Feedback Phase

At the end of the experiment we ask the participants for personal feedback. At first, we are interested in how the difficulty levels of the different tasks were perceived. For this purpose we ask the test persons to classify the three tasks into the categories *very easy*, *easy*, *difficult* and *very difficult*. In case there are many successes or many failures in the result set, we can check whether some tasks are perhaps too easy or too difficult and thus not appropriate in relation to the preparation described above.

Apart from that, the subjects should evaluate the following labeled statements formulated by us by assigning the statements to the categories *strongly disagree*, *disagree*, *agree* and *strongly disagree*:

S1: I felt like I was in the same room with my team member.

S2: I felt I could communicate and interact well with my team member.

S3: I felt working on Task 1 as a team was helpful.

S4: I felt working on Task 2 as a team was helpful.

S5: I felt working on Task 3 as a team was helpful.

S6: I consider the VR mode of ExplorViz to be suitable for analyzing software.

S7: I consider the VR mode suitable for working in a team.

Firstly, we are interested in whether, despite the spatial separation, interaction and communication with the team partner was perceived as intuitive and natural (see Statements 1 and Statement 2). On the other hand, we are interested in whether the collaboration was actually a help for the different task (see Statement 3, Statement 4 and Statement 5). Finally, we want to know the suitability of the VR extension for software analyzis as well as for collaboration (see Statement 6 and Statement 7). We deliberately do not use a neutral position as a possible answer in order to filter out as many tendencies as possible.

Furthermore, we ask a free text question, which features of the VR extension are considered helpful for collaborative use and which are not. In addition, we provide room for general feedback and comments to generate lessons and inspiration for future work and experiments.

6.4 Results

6.4.1 Test Subjects

To recruit participants for our experiment, we conducted a Doodle[7] survey. Participation was voluntary and without any consideration. 24 people participated in total, so we were able to form twelve teams. nine participants were students, eight participants were researchers and seven participants had their field of work in industry. The average age was 32.9 years.

In the following, we present further information about the teams and the test persons that we gained on the basis of the questionnaire in the preparation phase. We first transfer the various answer options from the questionnaire into a numerical system and then determine the arithmetic mean for the individual questions. This will give us an impression of the average preconditions under which a test person or a team stood. Table 6.4 shows the results of the questions regarding the team relationship, Table 6.3 shows the corresponding mapping for the computation.

Similarly, Table 6.6 shows the results of the questions regarding technical knowledge, Table 6.5 shows the underlying mapping.

[7] https://doodle.com/de/

Table 6.3. Integer mapping for questions regarding team relationship

Question ID	Category	Value
R1	not at all	0
	barely	1
	well	2
	very well	3
R2	never	0
	sometimes	1
	often	2

Table 6.4. Results of questions regarding team relationship

Question ID	Mean
R1	1.75
R2	1.13

Table 6.5. Integer mapping for questions regarding technical knowledge

Category	Value
None	0
Beginner	1
Intermediate	2
Advanced	3
Expert	4

Table 6.6. Results of questions regarding technical knowledge

Field ID	Mean
E1	2.88
E2	2.33
E3	2.17
E4	1.92
E5	1.83
E6	2.38
E7	1.63
E8	1.00
E9	1.08
E10	0.46

6.4.2 Assignment Phase

During the execution of the assignment phase, all teams without exception provided a solution proposal for each task. Table 6.7 shows the correctness of the answers in absolute values for each task.

Table 6.7. Results of the assignment phase

Task ID prefix	Task ID	Correct	Incorrect
A1	A1.1	11	1
	A1.2	12	0
	A1.3	12	0
A2	A2.1	10	2
	A2.2	12	0
	A2.3	11	1
A3	A3.1	8	4

6.4.3 Feedback Phase

In the feedback phase we first asked about the perceived level of difficulty. Table 6.9 shows the average values for the different tasks based on the mapping from Table 6.8.

Table 6.8. Integer mapping for perveived level of difficulty

Category	Value
very easy	0
easy	1
difficult	2
very difficult	3

Table 6.9. Results on perceived level of difficulty

Task ID	Mean
A1	0.67
A2	1.08
A3	1.21

The test persons also evaluated statements formulated by us. The summarized ratings of the statements are shown in Table 6.11. For the calculation we have used the mapping from Table 6.11. Here we have chosen integer values around zero, so that a positive value indicates tendency to agree with the statements and a negative value indicates tendency to disagree with them.

Table 6.10. Integer mapping for feedback statements

Category	Value
strongly disagree	-2
disagree	-1
agree	1
strongly agree	2

Table 6.11. Results of feedback statements

Statement ID	Mean
S1	1.50
S2	1.42
S3	0.58
S4	0.92
S5	1.00
S6	1.21
S7	1.21

The following criticism could be derived from the free text questions. On the one hand, many test subjects found the selection of entities very helpful, but it was often noted that the selection and differentiation of communication lines was a challenge. Furthermore, the hierarchical structure of the package visualization was rated to be very helpful for understanding the application, although there were complaints when reading the labels, as they were mostly abbreviated with dots. Another point of criticism was the performance. Especially opening and closing packages would lead to lagging of the application.

Additional features for the VR extension were also suggested, for example an option to open all packages simultaneously. Another suggestion was to make the direction of communication recognizable in the visualization independent of the infoboxes. Furthermore, it was suggested to integrate a whiteboard into the virtual space in order to record and compare certain information, e.g., information from infoboxes.

We also received feedback on the course and execution of the experiment. On the one hand, the interactive tutorial was judged to be very helpful, since one has to get used to the controls. Furthermore, it was noticed that the tasks were well and precisely explained. One point of criticism was that the fact that there were two people in each room communicating via Discord led to temporary double listening.

6.5 Discussion

In this section we will use the results of the experiment to answer the research questions formulated at the beginning of this chapter.

6.5.1 Are the Newly Added Features a Gain for the Use of the VR Extension?

We would rate the display of the key mapping (Feature F1) as very helpful for learning and using the program. For one thing, it ensures that several steps in the tutorial were self-explanatory so that we could reduce the time needed to get used to the program. On the other hand, the key mapping was used frequently by the test persons during the assignment phase, which was observable as we rarely had to intervene in case of control problems.

The selection of opened components as well as communication lines (Feature F2) was rated as very helpful in the received feedback. During the assignment phase, we further noticed that the selection function was frequently used to draw attention of the team member to certain entities. However, the feedback also revealed that a noticeable number of subjects had problems in picking communication lines with the controller.

The information range was increased by adding infoboxes for communication lines (Feature F3). Since we ask in particular about the number of requests or the average response time for Task A2.2 or Task A2.3, using the info boxes was essential for the complete solution of these tasks. Table 6.7 shows that Task A2.2 was solved correctly by 12 teams and Task A2.3 by 11 teams. In addition, Task A2 was given an overall difficulty level of 1.08, i.e., tending to be easy (see Table 6.9). These remarkably good results indicate that the infoboxes can be used sensibly in practice. However, the same point of criticism as with F2 exists here, namely that selecting the communication lines causes difficulties.

The highlighting of the entities according to the controller functions (Feature F5) was explained in the tutorial but was not mentioned explicitly in the feedback. In particular, however, we did not receive any negative feedback. This indicates that most likely no problems occurred and feature did not contradict the intuitive use of the program.

In summary, it can be said on the one hand, that the additional infoboxes have increased the information offer with regard to user-friendliness and, on the other hand, the possibilities of interaction have been increased by the extended selection function. In addition, the display of the key assignment supports an independent use of the VR extension. However, in the case of communication lines, the new features also pose problems for user-friendliness.

6.5.2 Is the VR Extension Suitable to Solve Complex Tasks in the Context of Static and Dynamic Software Analysis?

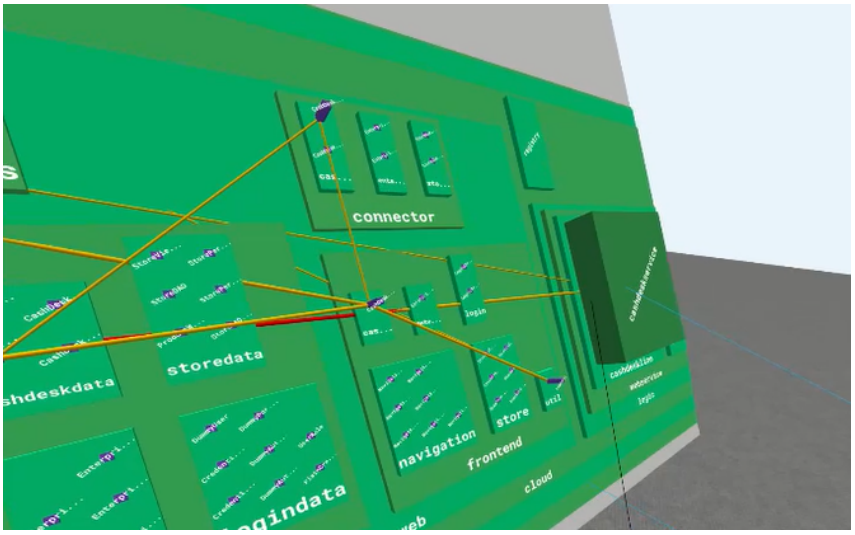
The Task A1 refers to the structure of CoCoMe, therefore the tasks include an analysis of the static factors of the software. Table 6.7 shows that almost all teams have completed all three tasks correctly. One team has processed Task A1.1 incorrectly. If one takes a look at the individual prerequisites of the team, it is noticeable that this team stated that it had neither knowledge of software architecture nor knowledge of static software analysis. However, there is an alternative explanation for the failure. As a reminder, in Task A1.1 we

asked for the top-level packages. These are the packages *cloud-web-frontend*, *cocome*, *org* and *de*. But the respective team only mentioned the first three packages. The reason may be that the package *de* was simply overlooked because of its small size. In the VR visualization, the size of the packages depends on the number of components they contain.

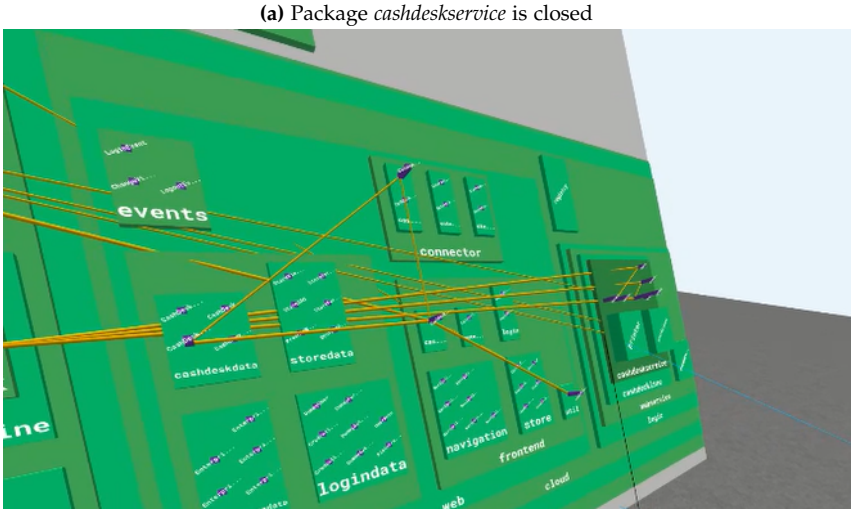
The Task A2 refers to the system load, hence the dynamics are analyzed. Again, the majority of the teams solved all tasks successfully but there were exceptions. There are two incorrect answers for Task A2.1. The goal of the task was to find the class with the most instances. An indicator for a high number is a high 3D class object in the visualization. This correspondence between the number of instances and the height of the 3D object was explained to the test persons in the tutorial. However, for the complete solution of the task it is necessary to compare the numerical values in the info box. This is because the relationship between the number of instances and object height is not proportional, instead there are few fixed object dimensions, each of which covers a certain range of values. Some teams have skipped the second step, the comparison of numerical values.

For Task A2.3, one answer is incorrect. As a reminder, in Task 2.3 we ask for the communication with the highest average response time. To solve the task, it is necessary to call up the info box for all the communication lines concerned and compare the numerical values. One of the main points of criticism in the feedback was that the communication lines were difficult to hit with the controller. This circumstance complicates the differentiation between the communication lines. This could be the reason for the failure.

Task A3 deals with the evolution of the software system and brings together static and dynamic analysis. Two thirds of the teams completed the task correctly, but there were also several wrong answers. In Task A3.1 we ask about classes affected by a change in the *BarcodeScanner* class. The correct answers from our point of view may differ depending on the interpretation, since the implementation of the classes is not disclosed. We accept the classes *CashDesk*, *Screenoverlay* and *TemplateFiller* as well as only the classes *CashDesk* and *Screenoverlay* as a solution, since the *BarcodeScanner* class calls the *TemplateFiller* but not the *TemplateFiller* calls the *BarcodeScanner*. Among the wrong answers there were two different answer types. One group only specified the class *Screenoverlay* as the solution, the only class to which the *BarcodeScanner* has a bidirectional relationship. Again, to be precise, depending on the implementation, the task could be interpreted differently. However, the other three teams only mentioned the unidirectional relationship between the *CashDesk* and the *BarcodeScanner* as a solution, although the directional relationship between *Screenoverlay* and *BarcodeScanner* is identical. During the assignment phase we were able to observe the reason for this. It can depend on the open status of components in the VR visualization whether the communication lines are visible. More precisely, the communication lines are higher compared to the floor of the 3D application when the packages containing the classes affected by the communication are open. Therefore, if many packages are closed, some communication lines may not be visible because they disappear in the remaining packages due to their low height (see Figure 6.3). We were not aware of this risk before the experiment.



(a) Package *cashdeskservice* is closed



(b) Package *cashdeskservice* is opened

Figure 6.3. Communication lines disappear due to their height

Apart from the success rate in the tasks, the degree of difficulty of the tasks was not rated as high (see Table 6.9). Interestingly, the tasks became more and more difficult from the perspective of the test persons, which is what we intended, especially with regard to task A3. Roughly rounded, however, the values are all around the value 1, i.e., the tasks tend to be easy. In addition, the test subjects tend to rate the VR extension with a value of 1.21 as suitable for analyzing software systems (see Statement S6 in Table 6.11).

In summary, it can be said that the test subjects achieved very good results in the Task A1 and Task A2. The difficulty, which was rated as low, and the positive feedback on Statement S6 confirm the assumption that the VR Extension is suitable for static and dynamic analysis. However, several problems have also been noticed. First, the test persons did not internalize all details in the short time, e.g., the fixed sizes in the instance-height correspondence of the classes. On the other hand, weaknesses in usability, e.g., the communication lines that are difficult to differentiate, threaten the correct task processing. Finally, the position of the communicaion lines depends on the open/closed status of components. This causes confusion in the visualization and leads to wrong results.

6.5.3 Is the VR Extension Suitable to Solve Complex Tasks in Collaboration?

For the experiment, we invited the test persons to the same virtual room with the aim of promoting and encouraging cooperation. The high level of agreement with Statement S1 shows that the test subjects had the perception that they were in the same room with their team member (see Table 6.11). During the assignment phase we observed intensive communciation among the test subjects and a frequent use of interaction mechanisms, e.g., the select function. The average agreement of 1.42 of the test persons regarding Statement S2 indicates that many test persons were in agreement that the VR extension supports communication and interaction among team members well. The high value for Statement Statement S7 also shows that the tool is generally perceived as suitable for teamwork.

With regard to the individual tasks, the results can be interpreted as follows. The results for Statement S3, Statement S4 and Statement S5 show that in all three task fields cooperation tends to be perceived as helpful (see Table 6.11). Interestingly, the values increase with each task. Thus, a correlation between the usefulness of the cooperation and the degree of difficulty of the tasks can be observed (see Table 6.9). This may suggest that increasing complexity motivates cooperation. An alternative explanation for the increasing values is that a progressive acclimatization to the VR environment as well as becoming acquainted with the respective team member encouraged a cooperation over time.

The high success in the tasks (see Table 6.7) indicates that the cooperation was not only perceived as helpful but was also effective in problem solving. Table 6.4 shows that the respective team members were well acquainted with each other on average and, as Question R2 confirms, had worked together occasionally before, which may have contributed to a successful cooperation. In total, three teams were not acquainted with each other. However, it is noticeable that two of the three teams solved all tasks correctly, i.e., achieved excellent results. A good level of familiarity is therefore not obligatory for effective cooperation in VR extension.

6.6 Threats to Validity

6.6.1 Test Persons

We had a total of 24 participants, so we had twelve teams. A higher number of participants would certainly lead to more representative results. However, in our experiment we were dealing with a very heterogeneous group of participants. For example, students, researchers, and people working in industry were equally represented.

6.6.2 Comparison Group

In our analysis we concluded that the cooperation was both perceived as helpful and effective in solving the problem. From this we can conclude that the cooperation in the VR extension works well in terms of program comprehension, but is not necessarily more effective than program comprehension in individual work. This is because we do not have a comparison group in which test persons solved the same tasks in isolation as it would be in a controlled experiment.

6.6.3 Hardware Configuration

The performance problems mentioned in the feedback are related to the hardware used. The usability of the VR extension for certain software landscapes as well as the usability of certain features depends on the computing power. Our results should therefore be considered in connection with the documented hardware configuration. However, we ensure that the same technical requirements were met by all teams. The use of different HMDs, which affects the tracking as well as the graphics, was equally set up for each team. Performance also depends on the rendering and layouting performance of ExplorViz which could be optimized.

6.6.4 Communication

During the assignment phase we communicated orally with the test persons via headsets. The experiment leaders read out the tasks and documented the proposed solutions. Misunderstandings in communication could have possibly led to false results, especially due to the confusion caused by the issue of double listening. A textual integration of the task into the VR extension as well as a textual input of the answers would have reduced the risk of misunderstanding. However, the experiment leaders have read the tasks several times and provided a guideline for clear communication when answering the questions. As we conducted the experiment with two experiment leader, they were able to verify mutual work steps.

6.6.5 Relevance of the tasks

To make the experiment more application-oriented, we employed CoCoME instead of a trivial example landscape. However, the tasks are specifically designed in a way that the participants use the different features of the VR extension and focus on cooperation. This may lead to the fact that the tasks are considered less relevant in the real context. Furthermore, real problems in the context of software analysis can be much more complex than the problems simulated in the experiment.

6.7 Conclusion

Based on the results of our experiment, it can be concluded that the VR extension is applicable for the collaborative work on complex problems in the context of static and dynamic software analysis. Hence, it is a useful contribution to program comprehension. The new features we have developed support this by providing additional information and extended possibilities of interaction. We have to relativize our conclusions insofar as the VR extension shows non-trivial problems in navigation and visualisation which negatively influence the use of the VR extension in certain use cases. These specific problems are the visibility of the communication lines that are dependent on the open status of the components and the differentiation of the communication lines. Thus, there is potential to improve the visualization of dynamic factors.

Chapter 7

Related Work

During our literature research we found an alternative approach to visualizing software for analysis purposes. Vincur et al. [2017] combine a 3D city metaphor and VR in their work, as does the ExplorViz VR extension, to explore object-oriented software systems. However, there are conceptual differences in the concrete visualization of the software components.

In the layout of Vincur et al. [2017] classes are represented by buildings. Methods are represented by the floors of buildings, where each floor consists of a certain number of blocks. The number of blocks is derived from any software metrics, for example lines of code (LOC). The shape, i.e., the height and width, of the class buildings is thus determined by the methods and the chosen software metrics. As a reminder, in the ExplorViz VR extension, the number of instances can be derived from the height of the 3D class objects. In the layout of Vincur et al. [2017], the varying shape of the class buildings shows further aspects depending on the chosen metrics. In a LOC based visualization, for example, thin and tall buildings can be identified as interfaces and buildings that are thin at the top can be identified as abstract classes (see Figure 7.1).

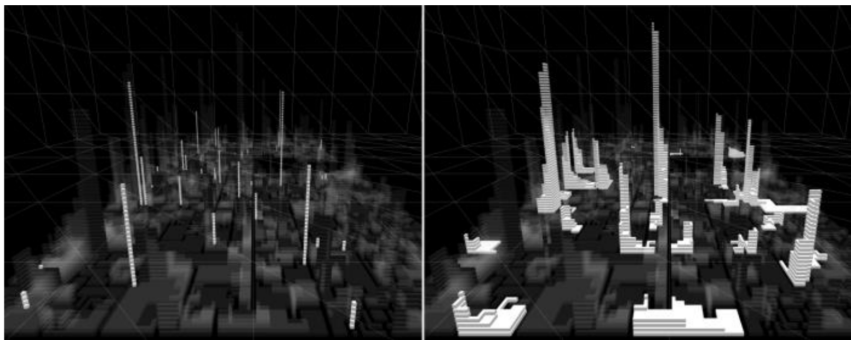


Figure 7.1. Highlighted interfaces (left) and abstract classes (right) [Vincur et al. 2017]

The position of the classes is derived algorithmically from the coupling of the classes. The packages are represented as districts. This shows a difference in perspective to the ExplorViz VR extension. In the layout of Vincur et al. [2017] all objects of the system are constantly visible but the model can be scaled to explore details. The ExplorViz VR extension, on the other hand, allows to choose the level of abstraction according to individual needs through a nested visualization of the package structure.

The ExplorViz VR model visualizes both static and dynamic aspects of the analyzed software system. According to Diehl [2007], however, besides the structure and behavior of software, evolution, i.e., the development process of the software, is a relevant aspect of software visualization. The layout of Vincur et al. [2017] integrates the evolutionary aspect by displaying author objects, which are related to model components on the basis of recent contributions. By selecting commits provided by the version control system Git[1], a certain time period of the software can be observed here.

In contrast to the abstract model of the ExplorViz VR extension, the VR visualization of Vincur et al. [2017] also integrates implementation details by displaying the source code of classes and methods in the VR environment.

In our experiment we used VR as a setting for the visualization of software systems. However, there are also many conservative visualization approaches using the standard desktop. This raises the question whether the two approaches differ in their use with respect to software visualization and whether immersion by means of VR is advantageous. With regard to this question we have come across different results.

In a controlled experiment with 20 participants, Rüdel et al. [2018] measure effectiveness and efficiency in orientation and navigation tasks and compare the results with respect to the visualization approach used, desktop or VR. The layout used for the visualization is EvoStreets, a layout to visualize evolving software systems using the city metaphor [Steinbrückner 2013]. Contrary to their hyphothesis, Rüdel et al. [2018] conclude that the subjects with VR headsets did not have it easier to gain spatial orientation.

Steinbeck et al. [2019] conduct a similar study comparing orthographic visualization using mouse and keyboard, 2.5D visualization using mouse and keyboard, and VR visualization using head-mounted displays. In a controlled experiment, subjects were to analyze Java systems based on EvoStreets [Steinbrückner 2013]. In contrast to the work of Rüdel et al. [2018], the focus here is less on orientation than on correctness. The results show that visualization in 2.5D or by means of VR did not lead to higher success in terms of correctness compared to orthographic visualization [Steinbeck et al. 2019]. In addition, they observed no differences in time and correctness of problem solving between the subjects using the 2.5D visualization and those using the VR visualization.

In another study, Romano et al. [2019] perform a controlled experiment in which the subjects are asked to perform program comprehension tasks either with the integrated development environment Eclipse[2] or with visualization tools on a standard desktop or in VR. The results show that the use of a visualization tool significantly increases the correctness of the tasks and the use of VR compared to desktop visualization significantly increases the speed of task solving [Romano et al. 2019].

An alternative approach to conservative desktop visualization but also to VR is visualization using Augmented Reality. In the work of Merino et al. [2019] the Microsoft HoloLens[3] is used to provide a dynamic visualization of the system's continuous per-

[1] https://git-scm.com/
[2] https://www.eclipse.org/
[3] https://www.microsoft.com/de-de/hololens

formance to developers on the basis of a city metaphor. Their results show that the city visualization is applicable to analyze dynamic performance data of large software systems [Merino et al. 2019].

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis we analyzed the ExplorViz VR extension in terms of usability and extended it with some features. Thereby we increased the information offer by additional infoboxes, expanded the possibilities of interaction by extending the selection function, and finally supported an independent use of the tool by integrating the key mapping.

Based on the improved VR extension, we conducted an experiment in which twelve teams worked on different tasks concerning program comprehension. The high success rates in solving the tasks as well as the overall approving feedback of the test subjects confirm the assumption that the VR extension is suitable for collaboratively analyzing both static and dynamic aspects of software systems. However, the experiment also revealed problems in navigation and layout. The selection as well as differentiation of communication lines was perceived as problematic by many test persons, and the dependence of the visibility of communication lines on the open status of components may lead to confusion and wrong results.

In conclusion, the VR extension provides a promising approach for collaborative program comprehension using VR, but also brings potential for further improvement.

8.2 Future Work

In Chapter 3 we have evaluated the VR extension in terms of improvement and extension potential. As previously mentioned, we did not implement all of the elaborated features but only a subset of features that are most relevant for our experiment. In future work the VR extension could be extended by these features. Especially fixing the freeze problem during (re)rendering (Feature PF7) would promote an unaffected VR experience. Furthermore, the visualization of traces (Feature PF11) would increase the expressiveness of the model. An interactive tutorial (Feature PF8) would enable independent familiarization.

Apart from the features above, the experiment has revealed even more optimization needs. The navigation and layout problems related to communication lines require appropriate solutions.

Furthermore, a 3D database visualization was developed for ExplorViz, which is also based on the city metaphor [Zirkelbach and Hasselbring 2019]. An application of VR could

possibly be useful for this to gain a system and program comprehension.

As previously stated, the VR extension is currently based on the outdated WebVR API. Instead, the WebXR[1] standard should be implemented in the long term.

Additionaly, in this thesis we have presented an approach to exploring the effectiveness of collaborative processing of program comprehension tasks using VR. Based on this approach, further experiments can be conducted to investigate alternative scenarios in the context of software analysis. It would also be interesting to conduct a controlled experiment in which the comparison group works on the tasks in isolation to find out whether collaboration is more effective than individual work.

[1] https://immersiveweb.dev/

Bibliography

[Brück 2020] J. Brück. Thesis Artifacts for: Collaborative Program Comprehension based on Virtual Reality (Apr. 2020). (Cited on pages 30, 34)

[Coates 2019] G. Coates. Program from Invisible site-a virtual sho, a multimedia performance work presented by George Coates Performance Works (Mar. 2019). (Cited on page 3)

[Desai et al. 2014] P. R. Desai, P. N. Desai, K. D. Ajmera, and K. Mehta. A Review Paper on Oculus Rift-A Virtual Reality Headset. *International Journal of Engineering Trends and Technology* 13.4 (July 2014), pages 175–179. (Cited on page 3)

[Diehl 2007] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Jan. 2007. (Cited on page 46)

[Fittkau et al. 2015] F. Fittkau, A. Krause, and W. Hasselbring. Exploring Software Cities in Virtual Reality. In: *Proceedings of the 3rd IEEE Working Conference on Software Visualization*. Bremen, Germany, Sept. 2015, pages 27–28. (Cited on page 5)

[Fittkau et al. 2017] F. Fittkau, A. Krause, and W. Hasselbring. Software landscape and application visualization for system comprehension with ExplorViz. *Information and Software Technology 87* (2017). (Cited on pages 1, 3–5)

[Hansen 2018] M. Hansen. Collaborative Software Exploration with the HTC Vive in ExplorViz. Bachelor's thesis. Department of Computer Science, Kiel University, Germany, Sept. 2018. (Cited on pages 1, 6, 7, 9, 27)

[Häsemeyer 2017] T. Häsemeyer. Kollaboratives Erkunden von Software mithilfe virtueller Realität in ExplorViz. Bachelor's thesis. Department of Computer Science, Kiel University, Germany, Sept. 2017. (Cited on page 5)

[König 2018] D. König. Collaborative Software Exploration with the Oculus Rift in ExplorViz. Bachelor's thesis. Department of Computer Science, Kiel University, Germany, Sept. 2018. (Cited on pages 1, 6, 9, 27)

[Mehrfard et al. 2019] A. Mehrfard, J. Fotouhi, G. Taylor, T. Forster, N. Navab, and B. Fuerst. A Comparative Analysis of Virtual Reality Head-Mounted Display Systems. *ArXiv* abs/1912.02913 (2019). (Cited on page 3)

[Merino et al. 2019] L. Merino, M. Hess, A. Bergel, O. Nierstrasz, and D. Weiskopf. PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness. In: *Proceedings of the Companion of the 2019 ACM/SPEC International Conference on Performance Engineering*. ICPE '19. Mumbai, India: Association for Computing Machinery, 2019, pages 13–16. (Cited on pages 46, 47)

[Rohr et al. 2008] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring. Kieker: Continuous Monitoring and on Demand Visualization of Java Software Behavior. In: *Proceedings of the IASTED International Conference on Software Engineering 2008*. Mar. 2008, pages 80–85. (Cited on page 3)

[Romano et al. 2019] S. Romano, N. Capece, U. Erra, G. Scanniello, and M. Lanza. On the Use of Virtual Reality in Software Visualization: The Case of the City Metaphor. *Information and Software Technology* (June 2019). (Cited on page 46)

[Rüdel et al. 2018] M. Rüdel, J. Ganser, and R. Koschke. A Controlled Experiment on Spatial Orientation in VR-Based Software Cities. In: *Proceedings of the 2018 IEEE Working Conference on Software Visualization, VISSOFT 2018, Madrid, Spain, September 24-25, 2018*. IEEE, 2018, pages 21–31. (Cited on page 46)

[Steinbeck et al. 2019] M. Steinbeck, R. Koschke, and M. O. Rudel. Comparing the EvoStreets Visualization Technique in Two-and Three-Dimensional Environments A Controlled Experiment. In: *Proceedings of the 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 2019, pages 231–242. (Cited on page 46)

[Steinbrückner 2013] F. Steinbrückner. Consistent Software Cities : supporting comprehension of evolving software systems. PhD thesis. BTU Cottbus - Senftenberg, 2013. (Cited on page 46)

[Sutherland 1968] I. E. Sutherland. A Head-Mounted Three Dimensional Display. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: Association for Computing Machinery, 1968, pages 757–764. (Cited on page 3)

[Van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE '12. New York, NY, USA: Association for Computing Machinery, 2012, pages 247–248. (Cited on page 3)

[Vincur et al. 2017] J. Vincur, P. Navrat, and I. Polasek. VR City: Software Analysis in Virtual Reality Environment. In: *Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2017, pages 509–516. (Cited on pages 45, 46)

[Zirkelbach and Hasselbring 2019] C. Zirkelbach and W. Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Technical report TR-1901. Department of Computer Science, Kiel University, Germany, Feb. 2019. (Cited on page 49)

[Zirkelbach et al. 2018] C. Zirkelbach, A. Krause, and W. Hasselbring. On the Modernization of ExplorViz towards a Microservice Architecture. *4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)* (Feb. 2018). (Cited on page 4)

[Zirkelbach et al. 2019a] C. Zirkelbach, A. Krause, and W. Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Technical report TR-1809. Department of Computer Science, Kiel University, Germany, Jan. 2019. (Cited on page 5)

[Zirkelbach et al. 2019b] C. Zirkelbach, A. Krause, and W. Hasselbring. Modularization of Research Software for Collaborative Open Source Development. In: *Proceedings of the Ninth International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2019), June 30*. July 2019. (Cited on pages 1, 3)

[Zirkelbach et al. 2019c] C. Zirkelbach, A. Krause, and W. Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Technical report TR-1902. Department of Computer Science, Kiel University, Germany, Apr. 2019. (Cited on pages 1, 4)