

# Collaborative Reengineering and Modularization of Software Systems

Dissertation

Christian Zirkelbach, M.Sc.

Dissertation  
zur Erlangung des akademischen Grades  
Doktor der Ingenieurwissenschaften  
(Dr.-Ing.)  
der Technischen Fakultät  
der Christian-Albrechts-Universität zu Kiel  
eingereicht im Jahr 2020

Kiel Computer Science Series (KCSS) 2021/4 dated 2021-09-06

URN:NBN urn:nbn:de:gbv:8:1-zs-00000378-a6

ISSN 2193-6781 (print version)

ISSN 2194-6639 (electronic version)

Electronic version, updates, errata available via <https://www.informatik.uni-kiel.de/kcss>

The author can be contacted via <https://www.explorviz.net>

Published by the Department of Computer Science, Kiel University

Software Engineering Group

Please cite as:

- ▷ Christian Zirkelbach. *Collaborative Reengineering and Modularization of Software Systems* Number 2021/4 in Kiel Computer Science Series. Department of Computer Science, 2021. Dissertation, Faculty of Engineering, Kiel University.

```
@book{Zirkelbach2021,  
  author   = {Christian Zirkelbach},  
  title    = {Collaborative Reengineering and Modularization of Software Systems},  
  publisher = {Department of Computer Science, Kiel University},  
  year     = {2021},  
  number  = {2021/4},  
  doi     = {10.21941/kcss/2021/4},  
  series   = {Kiel Computer Science Series},  
  note    = {Dissertation, Faculty of Engineering, Kiel University.}  
}
```

© 2021 by Christian Zirkelbach

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de> abrufbar.

# About this Series

The Kiel Computer Science Series (KCSS) covers dissertations, habilitation theses, lecture notes, textbooks, surveys, collections, handbooks, etc. written at the Department of Computer Science at Kiel University. It was initiated in 2011 to support authors in the dissemination of their work in electronic and printed form, without restricting their rights to their work. The series provides a unified appearance and aims at high-quality typography. The KCSS is an open access series; all series titles are electronically available free of charge at the department's website. In addition, authors are encouraged to make printed copies available at a reasonable price, typically with a print-on-demand service.

Please visit <http://www.informatik.uni-kiel.de/kcss> for more information, for instructions how to publish in the KCSS, and for access to all existing publications.

1. Gutachter: Prof. Dr. Wilhelm Hasselbring  
Christian-Albrechts-Universität zu Kiel  
Kiel
2. Gutachter: Prof. Dr. Matthias Riebisch  
Universität Hamburg  
Hamburg
3. Gutachter: Prof. Dr. Rainer Koschke  
Universität Bremen  
Bremen

Datum der mündlichen Prüfung: 19.05.2021



# Zusammenfassung

Softwaresysteme entwickeln sich stetig weiter. Anforderungsänderungen machen es für Entwickler unausweichlich Anpassungen vorzunehmen. Im Rahmen von Open Source Software existieren besondere Anforderungen, da jede Person Änderungen beisteuern kann. Besonders Forschungssoftware ist häufig nicht mit wart- und erweiterbaren Architekturen umgesetzt. Unzureichendes Wissen über die Struktur und das Verhalten von solchen Softwaresystemen und verwendeten Datenbanken kann weitere Probleme mit sich bringen. Daher stellt das Verständnis dieser Softwaresysteme eine wichtige Aufgabe dar, um die unausweichlichen Herausforderungen beim Durchführen von Änderungen zu meistern. Ansätze basierend auf alternativen Display- und Interaktionskonzepten können hierbei mithilfe von immersiven Benutzererfahrungen unterstützen.

In dieser Arbeit stellen wir drei Ansätze zur Unterstützung der Weiterentwicklung und insbesondere Verständnisses von Softwaresystemen mit verschiedenen Schwerpunkten vor. Unsere Hauptbeiträge sind (i) ein Ansatz, genannt *CORAL*, zur Durchführung von kollaborativem Reengineering und Modularisieren von Softwaresystemen, (ii) ein gestenbasierter und multi-user-fähiger virtueller Ansatz basierend auf Virtueller Realität, genannt *ExplorViz VR*, für die 3D Softwarestadtmeterapher und (iii) ein Ansatz zur Live Visualisierung von Datenbankverhalten, genannt *RACCOON*, zur Unterstützung des Verständnisses von Datenbanken in Softwaresystemen.

Eine umfassende Fallstudie zeigt, dass *CORAL* Reengineering- und Modularisierungsprozesse unterstützen kann. Des Weiteren demonstrieren mehrere Laborexperimente die hohe Benutzerfreundlichkeit sowie Effizienz und Effektivität unseres kollaborativen VR Ansatzes *ExplorViz VR* bei der Lösung von Programmverständisaufgaben. *ExplorViz* ist als Open Source Software unter [www.explorviz.net](http://www.explorviz.net) verfügbar. Zusätzlich stellen wir ein umfangreiches Paket unserer VR Evaluation zur Verfügung um die Nachvollziehbarkeit und Wiederholbarkeit unserer Ergebnisse zu ermöglichen.



# Abstract

Software systems evolve over their lifetime. Changing requirements make it inevitable for developers to modify and extend the underlying code base. Specific requirements emerge in the context of open source software where everybody can contribute and requirements can change over time. In particular, research software is often not structured with a maintainable and extensible architecture. Furthermore, often databases are employed for retrieving, storing, and processing application data. Insufficient knowledge of the actual structure and behavior of such software systems and related databases can entail further challenges. Thus, understanding these software systems embodies a crucial task, which needs to be addressed in an appropriate way to face inevitable challenges while performing software changes. Approaches based on alternative display and interaction concepts can support this task by offering a more immersive user experience.

In this thesis, we introduce three complementary approaches to support the evolution and particularly understanding of software systems in different aspects. Our main contributions are (i) an approach named *CORAL* for enabling collaborative reengineering and modularization of software systems, (ii) a gesture-based, collaborative, and multi-user-featuring Virtual Reality approach named *ExplorViz VR* for the software city metaphor, and (iii) a database behavior live-visualization approach named *RACCOON* for database comprehension of software systems.

An extensive case study shows that our *CORAL* approach is capable of supporting reengineering and modularization processes. Furthermore, several lab experiments demonstrate the high usability, and efficiency and effectiveness for solving comprehension tasks when using the visualization within our multi-user VR approach *ExplorViz VR*. All implementations are available as open-source software on [www.explorviz.net](http://www.explorviz.net). Additionally, we provide an extensive experimental package of our latest VR evaluation to facilitate the verifiability and reproducibility of our results.



# Preface

by Prof. Dr. Wilhelm Hasselbring

Reengineering and modularization of software systems is a non-trivial research field, and with his thesis Christian Zirkelbach has made original contributions to it. Christian Zirkelbach investigates how virtual reality can be leveraged for the collaborative analysis and comprehension of existing software systems. Specific contributions are the CORAL reengineering and modularization process, the multi-user 3D approach ExplorViz VR, and the RACCOON approach for live visualization of database behavior.

Highly innovative are the new techniques for collaborative 3D visualizations for improved program comprehension in teams and the new techniques for immersion into these 3D visualizations via topical virtual reality equipment.

Besides the conceptual work, this work contains a significant experimental part and a multifaceted evaluation. This engineering dissertation has been extensively evaluated with advanced student and lab experiments, based on a high-quality implementation of the ExplorViz tools. Furthermore, CORAL has been applied to ExplorViz itself.

This thesis is a good read and I recommend it to anyone interested in recent software reengineering, modularization and visualization research.

*Wilhelm Hasselbring  
Kiel, June 2021*



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation and Problem Statement . . . . .	3
1.2	Scientific Contributions . . . . .	6
1.3	Preliminary Work . . . . .	8
1.4	Document Structure . . . . .	17
<b>I</b>	<b>Foundations</b>	<b>21</b>
<b>2</b>	<b>Reengineering and Modularization</b>	<b>23</b>
2.1	Dynamic Analysis . . . . .	23
2.2	Forward, Reverse, and Reengineering . . . . .	24
2.3	Technical Debt . . . . .	25
2.4	Software Architecture . . . . .	26
2.5	Software Architecture Evaluation . . . . .	28
<b>3</b>	<b>Software Comprehension</b>	<b>29</b>
3.1	Definition and Motivation . . . . .	29
3.2	Software Visualization . . . . .	30
<b>4</b>	<b>Collaborative Software Engineering</b>	<b>39</b>
4.1	Definition and Motivation . . . . .	39
4.2	Model-based Collaboration Techniques . . . . .	41
4.3	Global and Multi-Site Collaboration . . . . .	43
<b>II</b>	<b>The Collaborative Reengineering and Modularization Approach <i>CORAL</i></b>	<b>47</b>
<b>5</b>	<b>Research Design</b>	<b>49</b>

## Contents

5.1	Research Methods . . . . .	49
5.2	Research Scope . . . . .	50
5.3	Research Plan . . . . .	50
<b>6</b>	<b>The CORAL Method</b>	<b>55</b>
6.1	Manual Analysis . . . . .	58
6.2	Tool-based Analysis . . . . .	62
6.3	Recommendation . . . . .	62
6.4	Execution . . . . .	64
6.5	Evaluation . . . . .	65
<b>7</b>	<b>Tool-based Analysis</b>	<b>67</b>
7.1	Program Comprehension through Software Visualization . . . . .	70
7.2	Improved Program Comprehension with Virtual Reality . . . . .	78
7.3	Distributed Multi-User VR Environments . . . . .	93
7.4	Database Visualization . . . . .	102
<b>III</b>	<b>Evaluation</b>	<b>113</b>
<b>8</b>	<b>Evaluation Overview</b>	<b>115</b>
<b>9</b>	<b><i>ExplorViz</i> VR Approach Evaluation</b>	<b>123</b>
9.1	Goals . . . . .	125
9.2	Case Study: Single-User VR Usability . . . . .	126
9.3	Case Study: Distributed Multi-User VR Environments . . . . .	142
9.4	Summary . . . . .	178
<b>10</b>	<b>CORAL Approach Evaluation</b>	<b>181</b>
10.1	Goals . . . . .	183
10.2	Case Study: <i>ExplorViz</i> . . . . .	183
10.3	Summary . . . . .	207



<b>11 Related Work</b>	<b>209</b>
11.1 Program Comprehension based on 3D Software Visualization	211
11.2 Virtual Reality . . . . .	212
11.3 Database Visualization . . . . .	216
11.4 Modularization and Reengineering . . . . .	217
11.5 Collaboration in Software Engineering . . . . .	219
<b>IV Conclusions and Future Work</b>	<b>223</b>
<b>12 Conclusions</b>	<b>225</b>
12.1 The <i>CORAL</i> Approach . . . . .	227
12.2 The <i>ExplorViz VR</i> Approach . . . . .	228
12.3 The <i>RACCOON</i> Approach . . . . .	229
12.4 Evaluation Results . . . . .	230
<b>13 Future Work</b>	<b>233</b>
13.1 The <i>CORAL</i> Approach . . . . .	235
13.2 The <i>ExplorViz VR</i> Approach . . . . .	236
13.3 The <i>RACCOON</i> Approach . . . . .	237
<b>V Appendix</b>	<b>239</b>
<b>A <i>ExplorViz VR</i> Evaluation - Questionnaires</b>	<b>241</b>
A.1 Single-User VR Usability Study - Questionnaire . . . . .	242
A.2 Distributed Multi-User VR Usability Study - Questionnaire . . . . .	247
A.3 Distributed Multi-User VR Applicability Experiment - Questionnaire . . . . .	254
<b>List of Figures</b>	<b>265</b>
<b>List of Tables</b>	<b>269</b>

Contents

**Listings** 271

**Bibliography** 273

## Chapter 1

# Introduction

Software is not limited by physics, like buildings are. It is limited by imagination, by design, by organization. In short, it is limited by properties of people, not by properties of the world. We have met the enemy, and he is us.

— Ralph Johnson

This chapter depicts an introduction to this thesis. In Section 1.1 the motivation for our research is described. Afterwards, we present our scientific contributions in Section 1.2. Then, we discuss preliminary work in Section 1.3. Finally, in Section 1.4, we provide a structural overview of this thesis.

## 1. Introduction

# Previous Publications

Parts of this chapter are already published in the following works:

1. Christian Zirkelbach. “Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems.” In: *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*. Hanover, Germany, 2017
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “On the Modernization of ExplorViz towards a Microservice Architecture.” In: *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*. Volume Online Proceedings for Scientific Conferences and Workshops. Ulm, Germany: CEUR Workshop Proceedings, Feb. 2018
3. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019
4. Christian Zirkelbach and Wilhelm Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Research Report 1901. Kiel University, Feb. 2019
5. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
6. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
7. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49

### 1.1 Motivation and Problem Statement

Large software landscapes often consist of a large number of heterogeneous systems, applications, and communication links [Pen93]. Although the landscape itself can appear to be consistent for a limited time, the included software systems are continuously evolving during their lifetime. Changing contexts, legal, or requirement changes [HRR+13] such as customer requests make it inevitable for developers to perform modifications to existing software systems. Open source software is based on the open source model, which addresses a decentralized and collaborative software development. Furthermore, open research software [Gob14; HCH+20] is available to the public and enables anyone to copy, modify, and redistribute the source code without costs and sometimes with only a few restrictions. In this context, where anyone can contribute code or feature requests, requirements can change over time and new user groups can appear. Although this development model features a lot of collaboration and freedom, the resulting software does not necessarily constitute a maintainable and extensible underlying architecture. Additionally, employed technologies and frameworks can become obsolescent or are not updated anymore. This causes a challenging task for developers during the development, especially when inexperienced collaborators like students are involved. Based on several drivers, like technical issues or occurring organization problems, many research and industrial projects need to move their applications to other programming languages, frameworks, or even architectures. Furthermore, technological modernizations depict an expensive and time-consuming task, because established software systems, especially monolithic ones, have often to deal with technical debt [Woo16]. Currently, a tremendous movement in research and industry constitutes a migration or even modernization towards an underlying architecture. Software architectures do not only influence quality attributes of a software system, but are also important in means of coordinating software projects [Her07].

A prevailing and frequently mentioned architecture constitute microservices, caused by promised benefits like scalability, agility and reliability [HS17]. Unfortunately, the process of moving towards a microservice-

## 1. Introduction

based architecture is difficult, because there are several challenges to address from both technical and organizational perspectives [FLM18; KH19] and design trade-offs [HB16]. Additionally, the development process is often unsuitable for performing collaborative software engineering which slows down the overall process and thus cannot develop its full potential. There are several reengineering and modularization approaches in research and practice. Unfortunately, there exists no reengineering and modularization approach to the best of our knowledge, which offers a holistic approach that supports developers in performing such processes. More precisely, there is a need for an approach that is capable of supporting modernization and modularization processes on the one hand but also focuses on collaborative aspects of the development process on the other hand.

Another crucial task within such processes embodies the comprehensibility of existing software systems. Large software landscapes can possibly contain hundreds of applications. One established way to perform this task is to apply software visualizations. More precisely, software visualization tools can significantly reduce the effort spent on system and program comprehension or maintenance related to software systems [CZv+09; BK01]. Furthermore, the importance of visualization increases, when reengineering and modularization tasks are performed. As these tasks constitute crucial steps in migration and modernization processes in software projects, it is reasonable to apply these tools not only for the above mentioned tasks, but also during the development of a software system. Based on software visualizations, it is possible to employ modern interactive and display methods to improve the comprehension process. Additionally, recent technologies like Virtual Reality (VR) offer the capability to develop more collaborative, immersive, and location independent approaches for comprehension purposes. Starting with the Oculus Rift DK1 head-mounted display (HMD), which was available at the end of 2013, the VR devices constituted a major step towards the consumer market. Based on this development, modern VR approaches became affordable and available for various research purposes. A similar development can be observed in the field of gesture-based interfaces, when Microsoft released their Kinect sensor in 2010 [Gar13]. A combination of both techniques offers new visualization and interaction capabilities for

## 1.1. Motivation and Problem Statement

newly created software, but can also improve the reverse engineering of existing software by means of immersive user experience [EPP15]. Furthermore, recent HMDs like the HTC Vive are often bundled with a pair of controllers, which facilitate the interaction with virtual environments even more. In the area of software engineering, exploring and understanding software systems is often handled through 2D or 3D visualizations [WL07; WWF+13]. Based on an in-depth 3D visualization and a more natural interaction, compared to a traditional 2D screen and input devices like mouse and keyboard, the user gets a more immersive experience, which benefits the comprehension process [FKH15a]. Recent publications indicate, that VR and Augmented Reality (AR) based approaches are utilized in several projects and can offer alternative, more immersive display and interaction concepts for software visualizations for several use cases [GKB+18; AAV+19b; AAV+19a; MBN18; MLS20]. To the best of our knowledge, there exists no collaborative VR based approach, which allows to explore software systems, visualized through the 3D software city metaphor, by using a HMD and gesture-based interaction that offers an immersive user experience for an improved comprehension process.

Furthermore, large software landscapes often consist of a large number of systems, applications, and communication links. Usually these systems employ databases, which provide well-defined interfaces for retrieving, storing, and processing application data. Databases are pivotal components in large software landscapes and are affected by the growing complexity and evolutionary progression of software systems. In combination with the steadily increasing amount of data, it is difficult to maintain a live overview of these software landscapes, database-related communication, and connected databases. This often leads to incomplete knowledge of the actual structure and behavior of related databases. Additionally, databases are often involved in performance issues within software systems [ZHC15]. Therefore, it is necessary to (i) monitor database queries invoked by applications, (ii) analyze connected databases, and (iii) present the gathered information in an appropriate form of visualization. Software landscapes are often visualized with UML-based deployment diagrams [OMG15] or modified versions to keep an overview of the landscape and included appli-

## 1. Introduction

cation systems [Flo16]. Databases by contrast are commonly represented by entity relationship diagrams [Che76]. To the best of our knowledge, there exists no visualization approach, which offers a software landscape facilitating the communication between applications and databases on the one hand, and the internal structure and actual usage of databases on the other hand. In [Zir17] we already described problems caused by missing or inappropriate database monitoring and visualization and presented a first sketch of a solution approach. Thus, there is a need for database visualization approaches, which support the comprehension process of databases and related communication for large software landscapes.

### 1.2 Scientific Contributions

Based on the motivation and problems mentioned in Section 1.1, this thesis makes the following three major scientific contributions (SC1 – SC3):

- SC1:** An approach named *CORAL* for enabling collaborative reengineering and modularization of software systems.
- SC2:** A gesture-based, collaborative, and multi-user-featuring VR approach named *ExplorViz VR* for the software city metaphor.
- SC3:** A database behavior live visualization approach named *RACCOON* for database comprehension of software systems.

The scientific contributions (SC1 – SC3) are described in the following.

#### **SC1: The *CORAL* Approach**

The first scientific contribution (SC1) of this thesis is an approach to support modernization and modularization processes in open source research projects named *CORAL*. The approach supports developers in performing such tasks by providing a holistic process, which guides them through a modernization and modularization. The approach consists of five, consecutive activities – starting with the analysis of an observed software system, divided into a *Manual Analysis* and a *Tool-based Analysis*, and ending with



the evaluation of an executed recommendation plan. By utilizing software visualization tools within the approach, we provide a better understanding of the observed software system and its underlying architecture. Additionally, we focus on the collaborative aspects of the process between developers and also aim for an improved development process. In order to evaluate our CORAL approach, we conducted a case study on the software visualization tool *ExplorViz*. More precisely, we performed multiple iterations of our approach while moving from a monolithic (referred to as *ExplorViz Legacy*) towards a microservice architecture. Thus, we showed that the approach is capable of supporting modernization and modularization processes of software systems.

### **SC2: The Collaborative Multi-User VR Approach** *ExplorViz VR*

The second contribution (SC2) depicts a gesture-based, collaborative, multi-user VR approach named *ExplorViz VR* for the software city metaphor. The approach complements the existing visualizations provided by *ExplorViz* and offers an alternative and immersive user experience for the system and program comprehension process. This fully collaborative, multi-user VR approach offers a combined landscape-level and application-level perspective. Thus, multiple users equipped with HMDs and related controllers are able to collaboratively explore and comprehend monitored software systems in VR. Based on several gestures mapped to related controllers the users can interact with the immersive software visualization and thus perform system and program comprehension tasks in teams. As software development takes more and more place on a globally scale, we also considered this aspect within the software and allowed a location-independent usage. Thus, developers located at different sites are able to utilize our approach for several tasks. The approach has been implemented as an extension to *ExplorViz* and is applicable within the *Tool-based Analysis* action as part of our CORAL approach. In order to evaluate the approach, we conducted three empirical experiments which verified its usability and collaborative applicability for system and program comprehension tasks especially in teams.

## 1. Introduction

### **SC3: Live Database Visualization with the *RACCOON* Approach**

The third contribution (SC3) is another software visualization approach based on the software city metaphor, named *RACCOON*. The approach supports performing comprehension tasks regarding the database behavior in software systems for developers and operators alike. *RACCOON* embodies a live visualization approach of databases and associated communication for large software landscapes. The employed visualization offers two different views — a landscape-level and a database-level perspective. The landscape-level perspective provides an overview of monitored applications and related databases. The database-level perspective reveals database schemas within a database, shows contained tables and relationships, and allows for the inspection of executed queries based on the monitoring information collected at runtime. Based on the visualizations, developers and operators both are able to investigate the actual usage of databases by applications and thus support such users in performance and comprehension tasks. Since this is still a work in progress, we have not realized an implementation and consecutive evaluation yet.

## **1.3 Preliminary Work**

In this section, we concisely summarize our 11 related publications and 14 supervised student works that have contributed to this thesis in ascending, chronological order. A more detailed description on listed bachelor's and master's theses, can be found in the corresponding works.

### **1.3.1 Related Publications**

- ▷ Christian Zirkelbach. “Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems.” In: *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*. Hanover, Germany, 2017

In this publication, we report on the lack of database monitoring in

### 1.3. Preliminary Work

long-living software systems. Furthermore, we (i) describe problems based on non-existing database monitoring in long-living software systems and (ii) propose an approach as a solution, which addresses the described problems, and supports developers and operators in performing evolutionary tasks alike.

- ▷ Robert Heinrich, Reiner Jung, Christian Zirkelbach, Wilhelm Hasselbring, and Ralf Reussner. “An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications”. In: *Software Architecture for Big Data and the Cloud*. Edited by Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim. Cambridge: Elsevier, June 2017, pages 69–89

In this publication, we successfully apply *ExplorViz* to a cloud-based software application in collaboration with the open-source research project *iObserve*. More precisely, we extract architectural information during runtime to support DevOps practices. Furthermore, we address differences between software development and operations and allow for phase-spanning usage of architectural models and provide system and program comprehension.

- ▷ Robert Heinrich, Christian Zirkelbach, and Reiner Jung. “Architectural Runtime Modeling and Visualization for Quality-Aware DevOps in Cloud Applications.” In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA 2017)*. 2017

In this work, we present a tutorial on modeling and visualizing software architectures in form of architectural runtime models to support quality-aware DevOps in cloud applications. The tutorial was held in context of the 14th IEEE International Conference on Software Architecture to share our findings and experiences with conference participants and give them the opportunity to expand their knowledge and skills on software architecture modeling, visualization, and analysis in development and operations.

## 1. Introduction

- ▷ Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “On the Modernization of ExplorViz towards a Microservice Architecture.” In: *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*. Volume Online Proceedings for Scientific Conferences and Workshops. Ulm, Germany: CEUR Workshop Proceedings, Feb. 2018

In this paper, we report on the initial modernization process and drivers of our open source research project *ExplorViz*. We describe the actual software version within the project and present how we solved a first modernization and handled occurring problems. Afterwards, we propose our modernized software system and point out the obtained benefits.

- ▷ Alexander Krause, Christian Zirkelbach, and Wilhelm Hasselbring. “Simplifying Software System Monitoring through Application Discovery with ExplorViz.” In: *Symposium on Software Performance 2018: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*. Nov. 2018

In this work, we report on our *ExplorViz* application discovery and monitoring management system (ADAMMS) to ease the monitoring configuration of *Kieker*. The key concept is to utilize a software agent that simplifies the discovery of running applications within operating systems. Furthermore, the ADAMMS properly configures and manages *Kieker* instances to monitor these applications. Finally, we conduct a first pilot study to evaluate the usability of our approach with respect to an easy-to-use application monitoring.

- ▷ Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019

In this technical report, we present our initial VR approach to explore software systems by using a HMD and two different gesture-based inter-

### 1.3. Preliminary Work

action devices. Hence, we achieve a more immersive user experience and natural interaction, which benefits the comprehension process. Our VR approach is integrated into *ExplorViz*, our tool for live trace visualization of large software landscapes. In order to emphasize the advantages, we apply our combined approach on a small software system.

- ▷ Christian Zirkelbach and Wilhelm Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Research Report 1901. Kiel University, Feb. 2019

In this paper, we present our live visualization approach of databases (RACCOON) and associated communication for large software landscapes. We describe the different visual representations and present first sketches of the proposed visualizations.

- ▷ Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019

In this technical report, we describe our ongoing modularization process and architecture of our open source research project *ExplorViz* towards a microservice architecture. An important aspect embodies a collaborative development process for both researchers and students. We describe the modularization measures and present how we solved occurring issues and enhanced our development process. Afterwards, we illustrate our modularization approach with our modernized, extensible software system architecture and highlight the improved collaborative development process. Finally, we present a proof-of-concept implementation featuring several developed extensions in terms of architecture and extensibility.

- ▷ Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. "Modularization of research software for collaborative open source development." In: *Proceedings of the Ninth International Conference on Advanced*

## 1. Introduction

*Collaborative Networks, Systems and Applications (COLLA 2019)*. June 2019

In this paper, we report on the ongoing improved modularization process and architecture of our open source research project *ExplorViz*. Based on the performed process we further improved our collaborative development process and achieved a better maintainability and extensibility.

- ▷ Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Leng, and Dan Kröger. “Microservice Decomposition via Static and Dynamic Analysis of the Monolith.” In: *Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2020

In this work, we present our experience with an approach that extends static analysis with dynamic analysis of a legacy software system’s runtime behavior, including the live trace visualization to support the decomposition into microservices. Overall, our approach combines established analysis techniques for microservice decomposition, such as the bounded context pattern of domain-driven design, and enriches the collected information via dynamic software visualization to identify appropriate microservice boundaries.

- ▷ Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49

In this paper, we report on the application of our *CORAL* approach to our open source research project *ExplorViz*. We explain how we performed the iterative modularization and reengineering approach with *CORAL*, applied measures, and describe how we solved occurring issues and enhanced our development process. Afterwards, we illustrate the application of our modularization approach and present the modernized, extensible software system architecture and highlight the improved collaborative development process. After the first iteration of the pro-

## 1.3. Preliminary Work

cess, we present a proof-of-concept implementation featuring several developed extensions in terms of architecture and extensibility. After conducting the second iteration, we achieved a first version of a microservice architecture and an improved development process with room for improvement, especially regarding service decoupling. Finally, as a result of the third iteration, we illustrate our improved implementation and development process representing an entire, separately deployable, microservice architecture.

### 1.3.2 Supervised Student Works

In the following, we list and briefly describe our supervised student works.

- ▷ Jan Witzany. “Instrumentierung von Android Anwendungen in ExplorViz.” Bachelor thesis. Kiel University, Sept. 2016

Witzany investigated the monitoring of mobile applications in *ExplorViz*. He described multiple approaches to instrument Android applications on the basis of Android Packages (APK). He extended the existing monitoring component and allowed the instrumentation of Android applications. For his evaluation he monitored and visualized a mobile medical application, which was developed by students.

- ▷ Maria-Anna Kandsorra. “Eye Tracking Based Experiments in ExplorViz.” Master thesis. Kiel University, May 2017

Kandsorra designed and realized eye tracking based experiments in *ExplorViz*. Her approach improved the usability and enhanced the existing experiment mode with the feature of eye tracking and recording the screen during an experiment. Furthermore, she conducted an experiment to evaluate her approach. Although the experiment results did not raise immediate correlations, the obtained eye tracking data still was valuable.

## 1. Introduction

- ▷ Timm Häsemeyer. “Kollaboratives Erkunden von Software mithilfe virtueller Realität in ExplorViz.” Bachelor thesis. Kiel University, Sept. 2017

Häsemeyer developed a preliminary version of our VR approach in form of an extension of *ExplorViz*. Starting with initial navigation and interaction concepts in addition to the visualization, he also sketched a first concept for the later developed multi-user approach. His evaluation, conducted with 14 participants, showed a good usability of the approach and led to a further development and finally multi-user capability of the approach.

- ▷ Matthias Möller. “Experiencing Software Landscapes using HCI in ExplorViz.” Bachelor thesis. Kiel University, Sept. 2017

Möller investigated alternative interaction concepts for *ExplorViz* based on Brain-Computer-Interfaces (BCI). Thus, he developed a proof-of-concept implementation with the Emotiv Insight device. In his evaluation, he focused on the user’s acceptance and applicability of the device for interaction purposes within our visualization.

- ▷ Felix Eichhorst. “Analyse der Microservices eines digitalen Marktplatzes mittels ExplorViz.” Master thesis. Kiel University, Oct. 2017

Eichhorst employed *ExplorViz* within the modernization process of a digital marketplace in the context of an industrial cooperation. He was able to present valuable findings to the main developers and made architectural recommendations for moving from a monolithic towards a microservice architecture.

- ▷ Alexander Krause. “Enterprise Application Discovery and Monitoring Management with ExplorViz.” Master thesis. Kiel University, Mar. 2018



### 1.3. Preliminary Work

Krause investigated the monitoring process of *ExplorViz* respectively *Kieker* within his thesis. Aiming to ease the monitoring configuration, he developed an extension of *ExplorViz*, which allows to configure the monitoring within the frontend on the basis of a monitoring agent. In his evaluation, he successfully demonstrated the usefulness of the improved configuration process such that we later integrated the extension in the core of *ExplorViz*.

- ▷ Josefine Wegert. “Visualizing Software Architecture Comparison of a Web-based Financial Application in ExplorViz.” Master thesis. Kiel University, May 2018

Wegert employed *ExplorViz* within the comprehension process of a financial application in the context of an industrial cooperation. Within her work, she developed an extension of *ExplorViz* to visually compare the software architecture of two different software versions. While applying the extension to two different release versions of a financial application she was able to present valuable findings to the developers and thus proved the applicability of the extension.

- ▷ Tim Hackel. “Architekturkonformitätsüberprüfung von Softwarelandschaften mittels ExplorViz.” Diploma thesis. Kiel University, Sept. 2018

Hackel’s thesis dealt with the aspect of software architecture conformance checking. Thus, he developed an extension of *ExplorViz* to visually compare a modeled and monitored software architecture of a software system. For his evaluation, he created a related scenario and let participants verify the usability and applicability of his approach.

- ▷ Malte Hansen. “Collaborative Software Exploration with the HTC Vive in ExplorViz.” Bachelor thesis. Kiel University, Sept. 2018 and Daniel König. “Collaborative Software Exploration with the Oculus Rift in ExplorViz.” Bachelor thesis. Kiel University, Sept. 2018

## 1. Introduction

Hansen and König continued the work of Häsemeyer and developed a first multi-user VR approach. Within their work, they utilized the HTC Vive respectively Oculus Rift and the related controllers for the realization. Based on WebSockets they exchanged relevant data and thus achieve a good extensibility and performance for their created real-time user environment. As a proof of concept, they conducted a usability evaluation with 22 subjects, which achieved good results.

- ▷ Florian Krippner. "Design und Implementierung eines Dashboards für ExplorViz." Bachelor thesis. Kiel University, Sept. 2019

Krippner investigated several concepts for a performance-oriented dashboard for *ExplorViz*. Afterwards, he developed a dashboard extension based on several performance and statistical metrics and conducted a usability evaluation. The results of the evaluation indicated a good usability of the approach.

- ▷ Daniel Teut. "Enabling Software Architecture Comparison with ExplorViz." Bachelor thesis. Kiel University, Sept. 2019

Teut's thesis dealt with the aspect of visual software architecture comparison, based on the work of Wegert. Based on his implementation, he improved the approach and allowed for a visual comparison of multiple versions of a monitored software system. For his evaluation, he conducted a usability study.

- ▷ Helge Müller. "A customizable and extensible Tutorial Framework for improved Usability in ExplorViz." Master thesis. Kiel University, June 2019

Müller developed a customizable and extensible tutorial framework in form of an extension of *ExplorViz*. The focus of the framework was

## 1.4. Document Structure

to provide a well-defined structure for tutorials, which could be used in experiments or for new users to explain several functionalities of *ExplorViz*. The usability of the framework was evaluated by questionnaires answered by participants after executing an example tutorial.

- ▷ Johannes Brück. “Collaborative program comprehension based on virtual reality.” Bachelor thesis. Kiel University, Apr. 2020

Brück continued the work of Hansen and König and improved the first multi-user VR approach in several usability and navigational aspects. Furthermore, one goal of this thesis was to investigate whether the VR extension is suitable for collaboratively solving complex tasks in the context of program comprehension. In order to evaluate its applicability, we conducted an experiment with 24 participants, in which software analysis tasks had to be solved by teams of two. The results showed that the VR extension is well suited for collaboratively analyzing both static and dynamic aspects of software systems.

## 1.4 Document Structure

This thesis consists of the following four parts:

- ▷ Part I describes the foundations for this thesis.
  - ▷ Chapter 2 presents terms and concepts regarding the reengineering and modularization of our *CORAL* approach required for the thesis.
  - ▷ Chapter 3 describes software comprehension and the underlying concepts and existing visualizations with a focus on related topics to our thesis.
  - ▷ Chapter 4 introduces the field of collaborative software engineering to our thesis.

## 1. Introduction

- ▷ Part II presents our three, complementary approaches (SC1 – SC3) for supporting the modernization and modularization processes of open source research projects on the one hand and software comprehension on the other hand.
  - ▷ Chapter 5 defines our research questions and employed research methods.
  - ▷ Chapter 6 describes our method, named *CORAL*, for supporting developers in modularizing and modernizing their software systems in an iterative manner (SC1).
  - ▷ Chapter 7 details our *Tool-based Analysis* action within our *CORAL* approach, including our VR approach *ExplorViz VR* (SC2) and database visualization approach *RACCOON* (SC3).
- ▷ Part III shows the performed evaluations for our presented approaches.
  - ▷ Chapter 8 provides an overview of our executed evaluations.
  - ▷ Chapter 9 describes the implementation and three conducted, consecutive evaluations of our VR approach *ExplorViz VR*.
  - ▷ Chapter 10 contains an evaluation of our *CORAL* approach in form of a case study applied to *ExplorViz Legacy* respectively *ExplorViz*.
  - ▷ Chapter 11 discusses related work.
- ▷ Part IV concludes the thesis.
  - ▷ Chapter 12 concludes the thesis and its evaluation results.
  - ▷ Chapter 13 presents future work.
- ▷ Part V depicts the appendix and contains the employed questionnaires for our *ExplorViz VR* evaluations in this thesis.

Finally, the back matter contains a list of figures, a list of tables, a list of listings, and last but not least the bibliography.





**Part I**

# **Foundations**





# Reengineering and Modularization

This chapter introduces terms and concepts regarding the reengineering and modularization processes for this thesis. Section 2.1 describes dynamic analysis in the context of software engineering. Afterwards, we define the terms forward, reverse, and reengineering in Section 2.2. Then, in Section 2.4 we give a brief introduction of software architecture. Finally, we describe the process of software architecture evaluation in Section 2.5.

## 2.1 Dynamic Analysis

Dynamic analysis comprises the process of analyzing a running program with focus to its deriving properties, which are valid at least for one or more executions of a program [Bal99]. This analysis step is often based on an instrumentation of a software or its compiled or interpreted program code. Compared to static analysis it provides advantages like detecting the most frequently utilized parts of the software and allows to make better assumptions about the behavior of the program [Bal99]. Furthermore, dynamic analysis, which extracts information from execution traces, can be used to overcome the limitations of static analysis [CDC11]. This way, the gathered information allows us to understand the software in a more detailed way. In this thesis, we utilize dynamic analysis to instrument software systems to perform system, program, and database comprehension based on software visualizations within our *CORAL*, *ExplorViz VR*, and *RACCOON* approaches.

## 2. Reengineering and Modularization

### 2.2 Forward, Reverse, and Reengineering

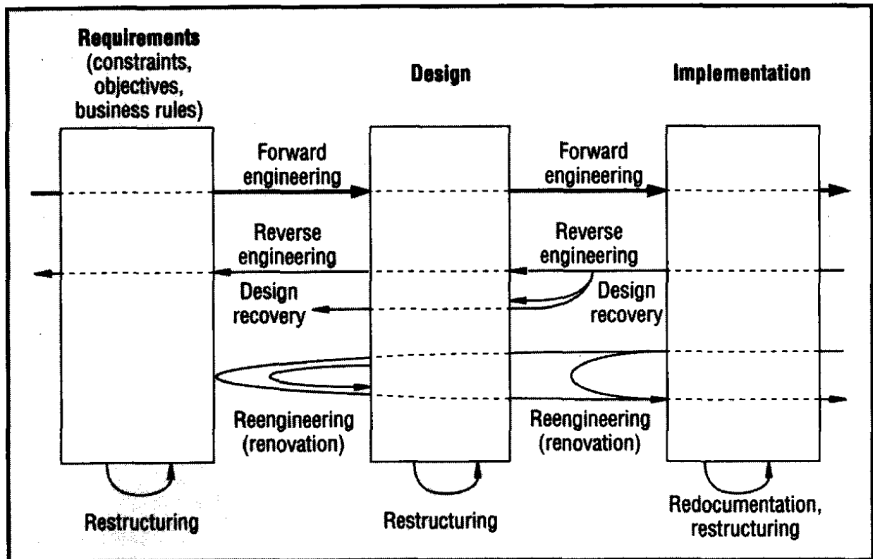


Figure 2.1. Reengineering and related processes in the context of software engineering [CC90].

Regarding software engineering, the term Reengineering was defined by Chikofsky and Cross II. [CC90] as “Reengineering, also known as both renovation and reclamation, is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent of the new form”. For an holistic overview of software reengineering we refer to [MQO18]. Figure 2.1 illustrates the differences between forward engineering, reverse engineering, and reengineering. The common process of forward engineering, within the software engineering domain, describes the process from high-level abstractions and designs towards a concrete physical implementation of a software system. In comparison, reverse engineering is quite the opposite, it characterizes the analysis of a previously engineered software system to identify its underlying architecture and individual software components,

which the software is composed of. Reverse engineering contains a wide range of methods and tools with reference to understand existing software and the ability to modify it [CD07]. Reverse engineering was defined as the process of “*analyzing a subject system to (i) identify its current component and their dependencies and (ii) to extract and create system abstractions and design information*” [CC90]. Many tools were developed in the past years to support this task with capabilities to explore, manipulate, analyze, summarize, and visualize software artifacts [MJS+00]. In the case of reengineering, often some form of reverse engineering followed by some form of forward engineering takes place [CC90]. In this thesis, we employ reengineering within our *CORAL* approach to perform a renovation and modularization based on the implementation, the design, and requirements of an observed software system.

## 2.3 Technical Debt

The concept of technical debt was first introduced by Cunningham [Cun92]. He described the term as a metaphor for a trade-off between writing clean code at higher cost and delayed delivery on the one hand, and writing messy code cheap and fast at the cost of higher maintenance efforts once it’s shipped on the other hand [Cun92]. A few years later, Kerievsky extended the metaphor towards software architecture and design by providing patterns [Ker05]. For an overview of technical debt, we refer to [KNO12; FBB+13; TAV13]. In Figure 2.2 an overview of the technical debt landscape is provided.

The figure shows a possible structuring of the technical debt landscape, divided into (external) visible elements like adding new features or fixing defects on the one hand, and invisible elements like architectural debt or low internal quality on the other hand. The invisible elements within this landscape can either be based on the realized source code, like anti patterns or code duplicates, or on the architecture like architectural or structural debt. Test debt is often caused by bad code coverage, missing edge cases, or even missing test cases. In this thesis, we identify technical debt within our *CORAL* approach for observed software systems. Furthermore, we analyze

## 2. Reengineering and Modularization

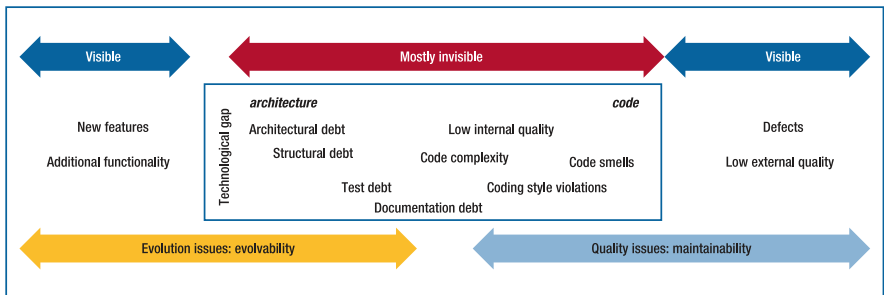


Figure 2.2. An overview of the technical debt landscape [KNO12].

technical debt to verify if the software quality has been improved after a performed iteration of *CORAL*.

## 2.4 Software Architecture

The underlying software architecture has a huge impact on developed software systems [Has18]. The architecture constitutes the foundation or structure of a software system. It massively affects how much effort is necessary to e.g., evolve, extend, maintain, and deploy a software system, especially in large software landscapes with a high number of systems and applications. Several definitions exist for the term software architecture within literature. In this thesis, we employ the definition based on the IEEE standard 1471 [MEH01; Sta17a], which describes software architecture as a general organization of a system, represented through its components, their relationship in-between and the environment, and finally the principles which apply for its design and evolution.

Software architectures are not only based on technical requirements. They are not optimal directly from the beginning, but instead act like chasing a moving target, which needs to be validated continuously [Sta17a]. Architectural design decisions also affect other aspects of the software engineering process and thus organizational processes. As a result, there is a continuous interaction of influence and counter-influence between them.

## 2.4. Software Architecture

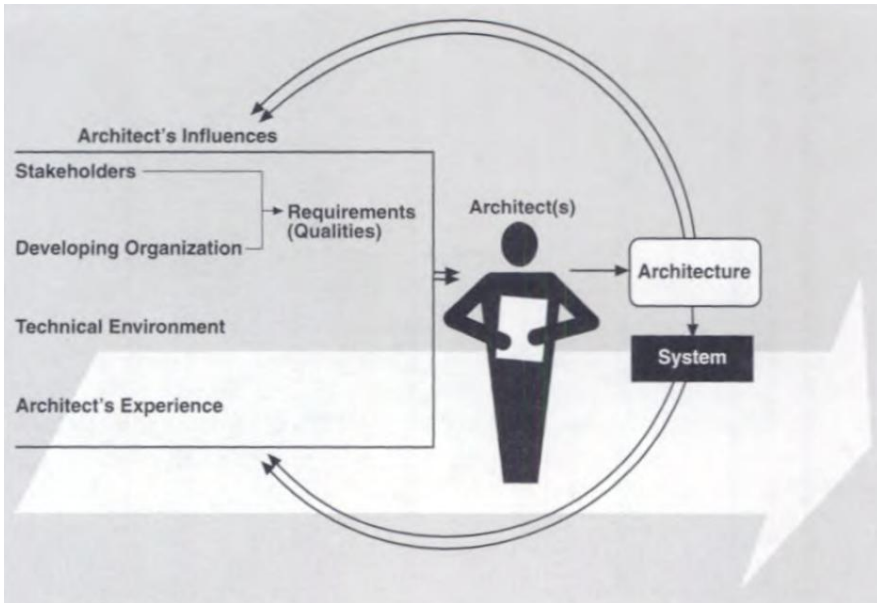


Figure 2.3. The Architecture Business Cycle [BCK03].

Bass et al. [BCK03] call this feedback mechanism “Architecture Business Cycle” (ABC), shown in Figure 2.3. The ABC shows how several aspects influence each other, which are briefly described in the following.

1. The architecture of a software system depicts its structure and thus forms the units of work, which affect the structure of the developing organization as well.
2. The architecture can affect the goals of the developing organization, by providing possible new opportunities based on the success of previously developed software systems. Thus, there is feedback from the software system towards the organization that builds it.
3. Successfully developing software systems can, in the case of software product lines, lead to changing customer requirements for the next

## 2. Reengineering and Modularization

software system.

4. Building software systems will affect the experience of involved software architects and developers.
5. Software systems can influence and change the software engineering culture, like J2EE at the beginning of the twenty-first century.

## 2.5 Software Architecture Evaluation

In order to assess a software architecture it is useful to perform a systematic software architecture evaluation based on a well-established method from the literature. In the past, several literature surveys regarding software architecture evaluation methods have been conducted in research [RG08; PS15] and industrial contexts [BN12]. For guidelines and approaches for evaluating software architectures, we refer to [CKK+03; BT06; BG09; Koz11; KN14]. Based on these reports and the requirements within our *CORAL* approach, we utilize the *Architecture Tradeoff Analysis Method* (ATAM) [KKC00] as our designated software architecture evaluation method within this thesis. It is a well-established method and has been utilized to evaluate several types of software architectures, also in the industrial context, over the past two decades. For case studies, guidelines, and best practices employing ATAM, we refer to [FHL02; BWS+06; GSI15; CPD+16; Sta17b; PDD+17; CAH+19].

# Software Comprehension

This chapter introduces the field of software comprehension and the underlying concepts and existing visualizations with a focus on related topics to our thesis. First, we present a definition of and motivation for software comprehension in Section 3.1. Afterwards, we describe comprehension via software visualization in Section 3.2.

## 3.1 Definition and Motivation

Understanding software is one of the most important activities within software engineering. Within the literature, there are several definitions of the term *Software Comprehension*. According to Deimel and Naveda [DN90], software comprehension is the process of “*taking computer source code and understanding it.*” Another, more suitable definition in our opinion provides Müller et al. [MWT95], by saying that “*Programmers use programming knowledge, domain knowledge, and comprehension strategies when trying to understand a program.*” He continues with “*...program understanding also tries to pattern match between a set of known plans (or mental models) and the source code of the subject software.*” In other words, software comprehension is a task which requires to develop mental models of a software system in several abstraction levels. This is necessary to (i) gain an overview of the software system and (ii) understand the software system in a more detailed way, from the software artifacts like source code, models like UML diagrams, or documentation. The comprehension of a software system does not only rely on the source code or programming languages, but also on the structure of the software system [LF19]. If a software engineer fails in understanding a software system, he is unable to perform modifications or extensions to

### 3. Software Comprehension

it in an appropriate and effective manner. Within the literature, there exist several software comprehension models that suggest, that programmers try to understand software using a bottom-up, top-down, or mixed comprehension approach. Thus, it is unlikely, that programmers rely on a single comprehension strategy exclusively [OBr03]. Software engineers need to switch between different comprehension processes to face this challenging task [MV95]. For a comprehensive overview of program comprehension and future directions, we refer to [Sie16].

Many studies and experiments have been conducted to evaluate approaches and techniques that aim to improve program comprehension processes. For a comprehensive overview of performed related research on program comprehension we refer to [SKS+17]. Recent research also addresses the remaining software familiarity of developers with a previously developed software system [KWF+19]. This is an important aspect within the software engineering process, because it comprises knowledge about the source code, design, architecture, and application of the software that has been forgotten over time.

## 3.2 Software Visualization

Visualization itself was defined by Card et. al [Car99] as *“the use of computer supported, interactive, visual representations of data to amplify cognition.”* According to Knight and Munro [KM99], software visualization (SV) is *“a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing software system under consideration”*. Furthermore, SVs are necessary for software developers to examine and comprehend their software systems since these are *“complex, abstract, and difficult to observe”* [PQ+06].

Within our multi-user VR approach *ExplorViz VR*, we utilize SV to visualize an observed software system based on dynamic analysis, i.e., by analyzing its runtime behavior, to support the comprehension process. To visualize the dynamic behavior of an observed software system, we need to perform four, consecutive steps.



1. Instrument a software system.
2. Collect runtime information.
3. Analyze the retrieved runtime information.
4. Visualize the analysis results.

Typical visualized aspects of a software system focusing on dynamic behavior are execution traces, resource consumption, or (object) interaction. For an overview of software visualization in general and with a focus on evolution and comprehension we refer to [War19; PQ+06; Die07; TC08; KBE+12].

### 3.2.1 Software City Metaphor

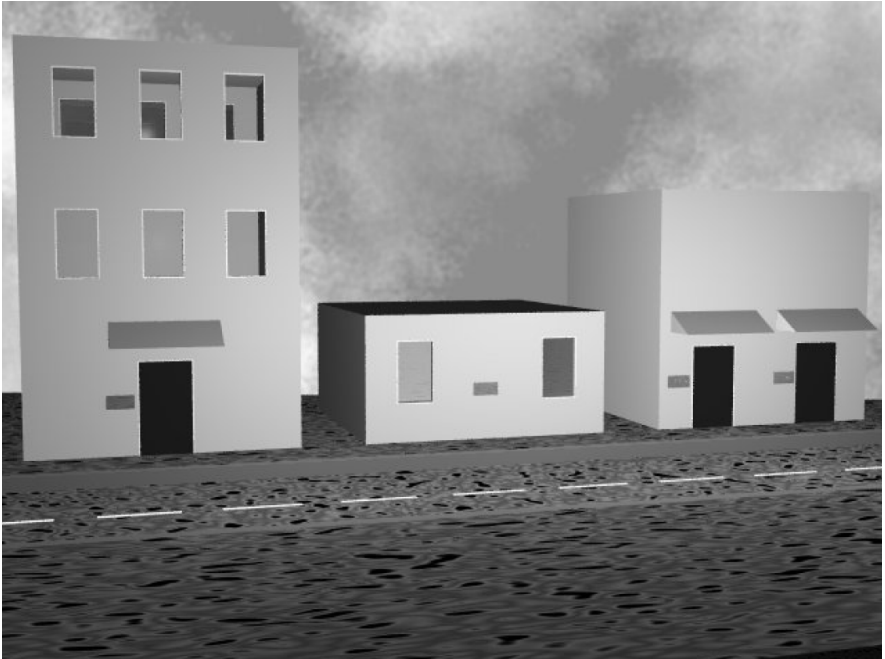
In the following, we describe the software city metaphor [KM99], which we employ within the visualization of our multi-user VR approach *ExplorViz VR*. We present three approaches based on the software city metaphor, which can be seen as milestone approaches towards our visualization.

#### 3.2.1.1 Software World

Knight and Munro [KM99] coined the term software city metaphor by presenting a software system based on the model of a real city, named Software World, in 1999. Within their approach, a software system is mapped towards a *world* and the directory structure of the system is visualized as a *country*, mapping packages in Java, within it. Each file within the directory structure embodies a *city* and each contained class is represented by a *district*. *Districts* contain *buildings*, which comply with methods of a class. Other Attributes of a class can be visualized as *urban items* within districts, e.g., gardens, parks, monuments, and streets, which act as a navigational support.

The approach offers a three-dimensional visualization, as shown in Figure 3.1. The height of a building maps to the lines of code of a method. Furthermore, buildings can have doors and windows, which map to the parameter count, respectively number of declared variables. The visibility

### 3. Software Comprehension



**Figure 3.1.** A street from Software World [KM99].

of a method determines the color of a building. Finally, method names are represented as small labels next to the related building.

#### 3.2.1.2 Code City

Based on several shortcomings like an insufficient support of the program comprehension process or interaction capabilities of Software World and successor approaches, Wetzel et al. [WL07; WL08] proposed their tool CodeCity. Their approach visualizes a software system also based on the software city metaphor, but uses a different mapping of software and visualization, especially for large software landscapes. Figure 3.2 shows an

## 3.2. Software Visualization

example visualization showing the software Findbugs,<sup>1</sup> a program which uses static analysis to detect bugs in Java code.

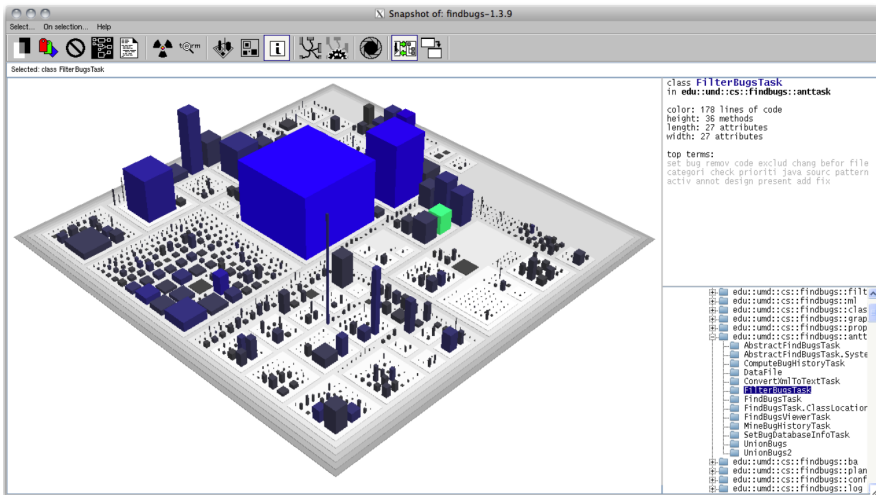


Figure 3.2. CodeCity: visualizing a structural overview of the software Findbugs [WLR10].

Basically, packages of a software system are represented as districts, which include classes in the form of buildings. The height of a building maps to the number of methods, while the width and length represents the number of attributes of a class. To easily spot patterns, the visualization provides several sub-types of buildings. Classes with a large number of methods but a small number of attributes result in tall and thin *skyscrapers*. In contrast, classes with only a small number of methods but a high number of attributes result in flat, large *parking lots*. *Office buildings* represent classes with a high functionality and a normal amount of attributes. Finally, classes with a low amount of functionality and a low number of attributes are visualized as *houses*. Furthermore, the approach utilizes colors to highlight further metrics, e.g., the probability of a god class. CodeCity provides an

<sup>1</sup><http://findbugs.sourceforge.net>

### 3. Software Comprehension

interactive and navigable exploration using the keyboard. Additionally, a user is able to zoom in on details of the city or to focus on one specific district by creating separate windows within the tool.

CodeCity has been evaluated in a controlled experiment, where Wetzel et al. [WLR10] compared the usage of their tool to using the state of the practice tools, i.e., Eclipse and Excel, for solving program comprehension tasks.

#### 3.2.1.3 *ExplorViz Legacy*

*ExplorViz Legacy* is a live trace visualization tool for large software landscapes [FWW+13; Flo16]. The tool focuses on system and program comprehension. It uses dynamic analysis techniques to monitor traces for large software landscapes and offers two different visualization perspectives, namely a landscape-level and an application-level perspective.

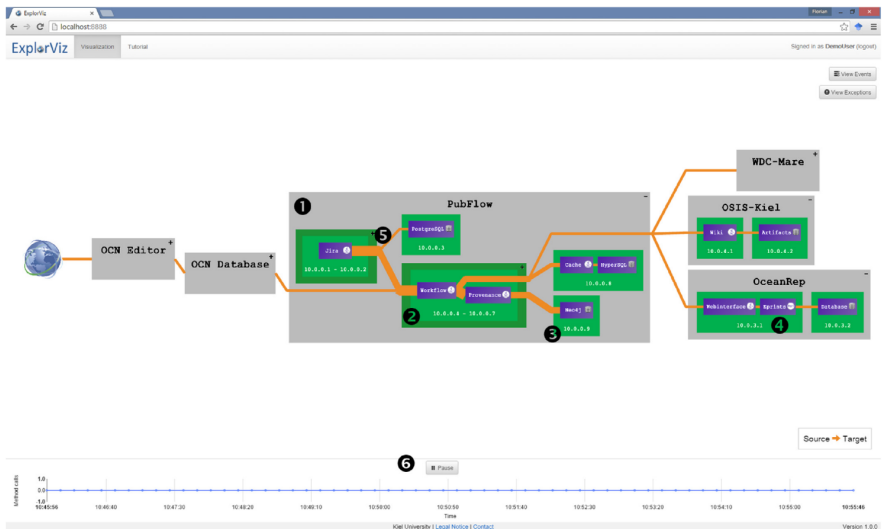
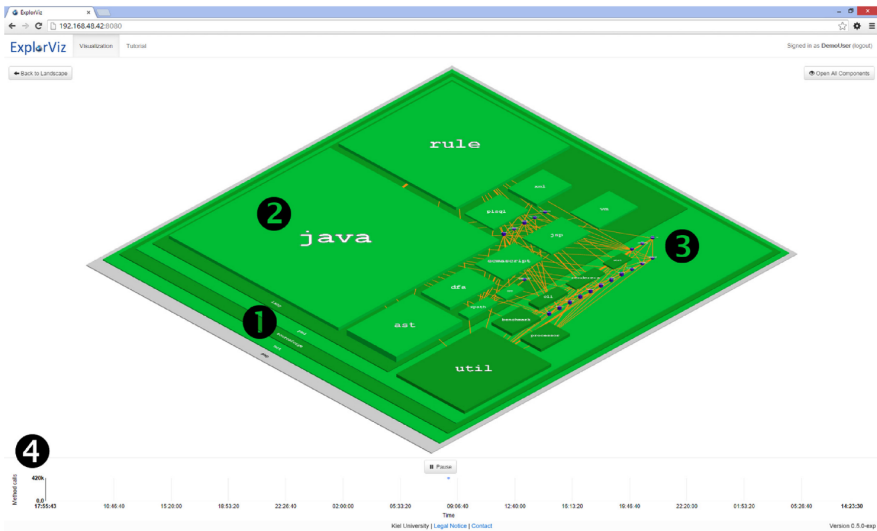


Figure 3.3. *ExplorViz Legacy*: Landscape-level perspective [Flo16; FRH15].

## 3.2. Software Visualization



**Figure 3.4.** *ExplorViz Legacy*: Application-level perspective [Flo16].

The landscape-level perspective, illustrated in Figure 3.3, employs a notation similar to data flow diagrams and provides an overview of the software landscape. The visualized software landscape consists of systems shown as gray boxes (❶) and included nodes colored in green (❷). Nodes with a similar configuration are bundled to node groups (❸). Applications, which are running on a specific node are visualized as purple rectangles (❹). Communication between applications is illustrated by orange lines (❺). The width of a line corresponds to the amount of communication. Furthermore, as the tool offers a live visualization capability, a time-shift option is provided, which allows a user to navigate back and forth in time (❻). Finally, it offers additional abstraction levels to highlight communication within the observed software systems.

The application-level perspective, displayed in Figure 3.4, utilizes the software city metaphor. Within this perspective, flat green boxes represent source code packages (❶). Additionally, green boxes on the top layer of the visualization (❷) hide internal details and can thus be opened or closed

### 3. Software Comprehension

interactively. The communication between classes is illustrated in the same way as in the landscape-level perspective by orange lines (③). Furthermore, the height of a class corresponds to the amount of active instances. Finally, the time-shift option, which is located at the bottom of the screen (④), allows the user to jump between several timestamps. This way, the provided 3D visualization is enhanced towards 4D.

The provided visualizations allow the user to interact with a generated 2D (landscape-level) respectively 3D (application-level) software model for enhanced program comprehension [FKH15a; FRH15; FKH15b].

#### 3.2.2 Virtual Reality

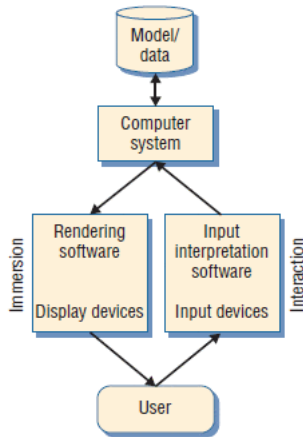
Virtual Reality (VR) is a concept and technology, which allows an immersive experience of several scenarios, such as entertainment, education and training, or low frequency and high-risk. First concepts of VR already came up in the 1960s, leading to a vision of a VR-like “*ultimate display*” presented by Ivan Sutherland [Sut65] in 1965. A few years later he discussed the first version of a tracked 3D head-mounted display (HMD) and its applicability [Sut68]. The term Virtual Reality was coined by Jaron Lanier [MT94] in the 1980s as founder of VPL Research, the first company to sell VR products. Additionally, he led the team that developed the first widely used software platform architecture for immersive VR applications.<sup>2</sup> The first hype for VR started in the 1990s and yielded several related applications and hardware including technological improvements [Bro99]. Unfortunately, VR could not establish itself due to several conceptual, technological, and economic hindrances. As a consequence, its successful entry to the consumer market was delayed until 2016 with the release of the Oculus Rift. In the same year Anthes et al. [AGW+16] presented an overview of state-of-the-art VR technology and compared several types of VR HMDs and their limitations. Furthermore, they described the (tremendous technological) advances as “*the second wave of VR.*” Since then, new applications and successor devices were released at a high pace until today.

According to Bowman and McMahan [BM07] hardware and software can play a role to determine the level of immersion a user perceives when

---

<sup>2</sup><http://www.jaronlanier.com>

## 3.2. Software Visualization



**Figure 3.5.** Human and virtual environment interaction loop [BM07].

using VR. More precisely, they can limit the perceived immersion of a user. An overview of the interaction between a user and the employed hardware and software in this context is shown in Figure 3.5. A VR system with relevant components within an interaction loop is presented. Within the loop, a computer system retrieves data from a model or database and creates a visualization first. Afterwards, a suitable software renders the VR visualization on a display device, which then creates an immersive experience for a user. The user in turn interacts with a VR system using input devices and thus sends feedback to the VR system.





# Collaborative Software Engineering

This chapter introduces the field of collaborative software engineering to our thesis. First, we present a definition of and motivation for collaborative software engineering in Section 4.1. Afterwards, we describe model-based collaboration techniques in Section 4.2. Finally, we describe collaboration in global and multi-site software engineering in Section 4.3.

## 4.1 Definition and Motivation

Collaboration depicts a central, inevitable aspect in each software engineering process [Whi07]. Whitehead et al. [WMG+10] state, that *“any software project with more than one person is created through a process of collaborative software engineering.”* Additionally, they present a more precise definition with *“Collaborative software engineering deals with methods, processes, and tools for enhancing collaboration, communication, and co-ordination (3C) among team members.”* Thus, collaboration embodies an important aspect within the software development process.

A well-known parable involving several blind men and an elephant as shown in Figure 4.1, originated in the ancient Indian subcontinent, motivates the collaboration characteristic from a different perspective. Within the story a group of blind monks, who have never met an elephant before, learn and conceptualize an elephant by touching it. Since every monk touches a different part of the elephant, they describe him based on their restricted experience. Thus, their descriptions of the elephant differ significantly. Large

## 4. Collaborative Software Engineering



**Figure 4.1.** Blind monks examining an elephant, an ukiyo-e print by Hanabusa Itchō (1652–1724).

software systems are like the elephant in this story and each software developer has his own point of view and understanding of it [WMG+10]. The major difference between the elephant and a software system is that a software developer cannot simply touch it and thus receives an understanding. Instead, software systems are built and particularly shaped by a collaborative development process, involving a large number of developers with different educational, cultural, social backgrounds, and experiences. In other words, software engineering collaboration is the mingling between the mental concept and understanding of a software system by each involved developer [WMG+10].

Another approach to address collaboration within software engineering depicts collaborative model-driven software engineering (CMDSE). CMDSE

## 4.2. Model-based Collaboration Techniques

is based on models and considers them as first-class citizens, which guides software development activities and other model-based tasks in the context of the software engineering process [FLM18]. For a recent overview of collaborative model-driven software engineering we refer to [FRM+18].

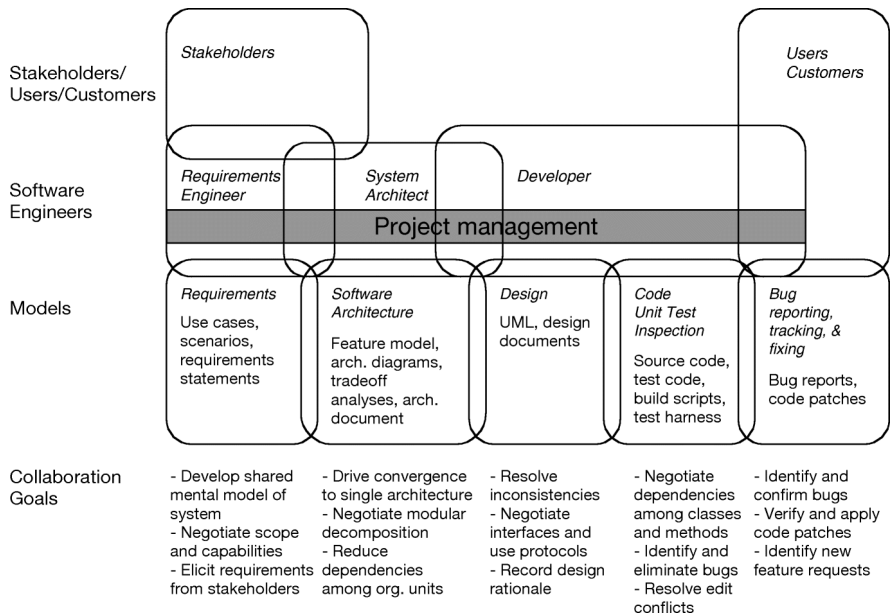
### 4.2 Model-based Collaboration Techniques

A wide range of model-based collaboration techniques, technologies, and tools have been developed to support collaborative work on software systems and projects [Whi07]. According to Whitehead [Whi07], tools supporting the collaboration in software engineering can be divided into four categories *Model-based Collaboration*, *Process Support Tools*, *Awareness Tools*, and *Collaboration Infrastructure*. *Model-based Collaboration* tools allow software developers to collaborate in several phases during the software development process. *Process Support Tools* cover and support the software development process or at least single phases of it. *Awareness Tools* aid developers by notifying them on actions performed by others to avoid conflicts. Finally, *Collaboration Infrastructure* provides a solid, platform independent framework, which guides data and control integration for a software system and project. In the following, we introduce *Model-based Collaboration Tools* and their influence towards the software engineering process.

#### 4.2.1 Model-based Collaboration Tools

An overview of model-based collaboration within this process is shown in Figure 4.2. Within the figure, rows represent different types of actors or models and columns represent different phases in the software development process. Additionally, overlaps between rectangles with rounded corners for types of actors represent collaboration. The actor software engineer involves the roles of a requirements engineer, a system architect, or a developer, depending on the respective phase. Based on this visualization, we describe the collaboration within the phases in the following.

## 4. Collaborative Software Engineering



**Figure 4.2.** Overview of model-driven collaboration [WMG+10].

### 4.2.1.1 Requirements

Within the requirement phase, software engineers and stakeholders need to develop a shared mental model of the system which should be built. An established way to do this is to use tools supporting the requirement analysis and negotiation between software engineers and stakeholders on the one hand, and using explicit model-based representations, e.g., UML [OMG15] (use case) diagrams, on the other hand.

### 4.2.1.2 Architecture

Choosing a and committing to a specific software architecture for a software system is a collaborative process even if technical or organizational demands or boundaries exist. Decisions within this phase impact the software

## 4.3. Global and Multi-Site Collaboration

systems in several aspects, like maintainability or extensibility. Thus, it is important to perform a modular decomposition and reduce dependencies among organizational units. Tools are able to support creating architecture-related documents and diagrams, which can also act as a specification and documentation. Even for performing a software architecture evaluation there exist collaborative approaches [MT05].

### 4.2.1.3 Design

The design phase affects the provided capabilities and features of a software system. Therefore, designing parts of the software system like components or interfaces depicts an essential task within the process. Software engineers have to collaboratively resolve inconsistencies, negotiate interfaces and use protocols, and especially document their design decisions. Within this field of work, several tools exist, which allow to collaboratively design, discuss, and document on the basis of UML diagrams.

### 4.2.1.4 Test

Testing and inspecting software systems and their source code is also an integral part of the software engineering process. Testing often involves users and thus requires a collaboration between them and the software engineers. Furthermore, often development teams and test teams need to work together to achieve a common purpose. Hence, it is necessary to offer collaborative tools, which support these aspects to negotiate dependencies among classes and methods, identify and eliminate bugs, and resolve occurring merge conflicts. Issue and bug tracking systems provide such capabilities and also facilitate the communication between software engineers and users, even after the software system has been released. Additionally, such systems can be employed to identify and submit new feature requests.

## 4.3 Global and Multi-Site Collaboration

Global software engineering (GSE) extends classical software engineering in terms of needing methodical and technical support to organize, man-

#### 4. Collaborative Software Engineering

age, and execute distributed software projects. According to Ebert et al. [EKP16] GSE, IT outsourcing, and business process outsourcing have shown growth rates of 10-20% per year since 2006. Although, approximately 20% of all globalization projects are canceled within the first year, and about 50% are terminated early [EKP16]. GSE adds further complexities, such as geographical, temporal, cultural, and linguistic distance to the software development process [Whi07]. Particularly, the distribution of software developers working on a software project has a significant impact on the development process and thus on the built software system. Furthermore, the global development emerges additional problems [OO00; BM02]. Differences between co-located and global software development have been described by Herbsleb [Her07]. Additionally, Khan and Keung performed a review of success factors and barriers for software process improvement in GSE in [KK16]. For a comprehensive overview of collaboration in GSE we refer to [YAW+19].

One solution for supporting GSE projects embodies the usage of collaborative development environments (CDE). CDEs provide a project workspace with a standardized tool set based on online services and cross-platform support. *ExplorViz Legacy* respectively *ExplorViz* and our presented VR approach *ExplorViz VR* within this thesis offer a special kind of CDE focusing on collaborative system and program comprehension of software systems. Thus, they facilitate multi-site and even global collaboration based on their web-based realization. For a recent comparison of collaborative development environments we refer to [LEP+10].







**Part II**

**The Collaborative  
Reengineering and  
Modularization Approach  
*CORAL***



# Research Design

In this chapter, we present our research design to face challenges regarding modernization and modularization in open source research projects described in Section 1.1. We start with presenting the employed research methods in Section 5.1. Afterwards, we describe the research scope including our research goals in Section 5.2. Finally, we explain in Section 5.3 how we address our research goals by presenting a research plan.

## 5.1 Research Methods

We base our research on the established Goal, Question, Metric (GQM) approach [BW84; SB99; SBC+02]. Additionally, we employ the following, additional research methods listed below. These follow, in the area of software engineering, approved research methods [WRH+12; JG12] in combination with empirical software engineering guidelines [KPP+02; KDJ04].

- ▷ Literature Review: We conduct a literature review to identify and review related concepts, models, and approaches.
- ▷ Proof-of-Concept Implementation: We develop proof-of-concept implementations in order to evaluate the approaches and their technical feasibility.
- ▷ Case Study: We conduct several case studies to further evaluate the approach and especially measure the extensibility and collaboration-oriented aspects of our approaches. Additionally, we work together with other open source research projects and industrial partners to verify our approaches in real environments.

## 5. Research Design

- ▷ Lab Experiment: We employ lab experiments in order to evaluate the approaches and verify their usability.

### 5.2 Research Scope

The scope of our research covers the support of reengineering and modularization processes of software in open source research projects. As described in Chapter 1, there is a need for comprehensive modernization and modularization approaches, which aid involved developers and ease the above mentioned processes of such projects in a collaboration-oriented manner. Thus, our main research goal (G1) is to provide a solution approach, which addresses both technical and organizational perspectives. In this context, investigating alternative visualization and interaction concepts for the software city metaphor is our second goal (G2). Our last goal (G3), concerns researching and applying alternative program comprehension approaches based on the software city metaphor.

These three goals are described according to the GQM approach in the following and map to our scientific contributions (SC1 – SC3) described in Section 1.2.

### 5.3 Research Plan

As described in Section 5.2, our main research goal is to provide a solution approach, which supports the modernization and modularization process in open source research projects and addresses both technical and organizational perspectives for involved software. Focusing on the goals, we define the following GQM-based [BW84; SB99; SBC+02] research plan:

#### **G1: Provide an approach to support modernization and modularization processes in open source research projects**

In open source research projects several drivers, like technical issues or changing customer demands, make it inevitable to perform modifications on employed software systems. Software changes are rarely an easy task,

because they are affected by a huge number of parameters. With a generic solution approach, we support the modernization and modularization processes of software in these open source research projects. Thus, the corresponding research question (Q1) concerning G1 is: *How to design and implement an approach to support modernization and modularization processes of software in open source research software projects?*

To answer this question, we will evaluate existing related approaches first. In detail, we focus on examining underlying software architectures, utilized software stacks, and employed development processes. Furthermore, we will design, develop, and evaluate a new approach for supporting developers in open source research projects with respect to our focus. Additionally, we will invest major effort to provide an extensible software architecture, which eases the process of evolution and maintenance. The ability to develop functional extensions or modifications to existing open source research projects plays an important role in our approach. Thus, these extensions are evaluated with the help of case studies. Our associated metrics are therefore:

- ▷ **M1.1** Number of different, existing supporting approaches.
- ▷ **M1.2** Number of proof-of-concept-implementations.
- ▷ **M1.3** Number of realized extensions.
- ▷ **M1.4** Number of conducted case studies.

#### **G2: Find and employ alternative display and interaction concepts for system and program comprehension using the software city metaphor**

The second goal of the thesis is to take a look at alternative display and interaction concepts and how can they be used in such processes in combination with the software city metaphor. Thus, the corresponding research question (Q2) concerning G2 is: *How to employ alternative display and interaction concepts for system and program comprehension using the software city metaphor?*

## 5. Research Design

To answer this question, we will take a look at modern, alternative display and interaction concepts, which are suitable in combination with the software city metaphor. Based on this, we will design and implement corresponding concepts focusing on enhancing software visualization in a collaborative manner. Afterwards, we will evaluate these approaches regarding their usefulness and usability with the help of case studies. Our associated metrics are therefore:

- ▷ **M2.1** Number of concrete, alternative representation forms.
- ▷ **M2.2** Number of realized, alternative display and interactions concepts.
- ▷ **M2.3** Number of conducted case studies.

### **G3: Research and evaluate alternative visual approaches for system and program comprehension**

In open source research projects, a huge number of changes is made to research software systems as the project evolves over time. Keeping track of changes is difficult, especially when changes are architectural or structural-based with the software. An attempt to address this problem is to employ alternative program comprehension approaches based on visualization or even visual comparison. Therefore, our third research question (Q3), which corresponds to G3 is: *Q3: How to research and evaluate alternative visual approaches for system and program comprehension?*

To answer this question, we will evaluate program comprehension approaches with respect to software visualization based on visual comparison. Afterwards, we will design and implement corresponding concepts focusing on enhancing software visualization in a collaborative manner. Finally, we will evaluate whether these concepts are beneficial for the comprehension process. Our associated metrics are therefore:

- ▷ **M3.1** Number of existing visual system and program comprehension approaches.

### 5.3. Research Plan

- ▷ **M3.2** Number of realized, visual system and program comprehension concepts.
- ▷ **M3.3** Number of conducted case studies.





# The CORAL Method

In this chapter, we present our *CORAL* method. The approach addresses problems regarding the modernization and modularization of open source research software in technical and organizational aspects. In detail, we describe how we employ the approach to achieve the goals presented in the preceding chapter. We propose a collaborative, tool-employing approach to support developers in modularizing and modernizing their software systems in an iterative manner. Our approach consists of five, consecutive activities – starting with the analysis of an observed software system and ending with the evaluation of an executed recommendation plan. Fig. 6.1 gives an overview of the approach in form of a UML activity diagram. The five activities (colored in gray) are *Manual Analysis*, *Tool-based Analysis*, *Recommendation*, *Execution*, and *Evaluation*. In the following, the activities are briefly described.

## 6. The CORAL Method

### Previous Publications

Parts of this chapter are already published in the following works:

1. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49

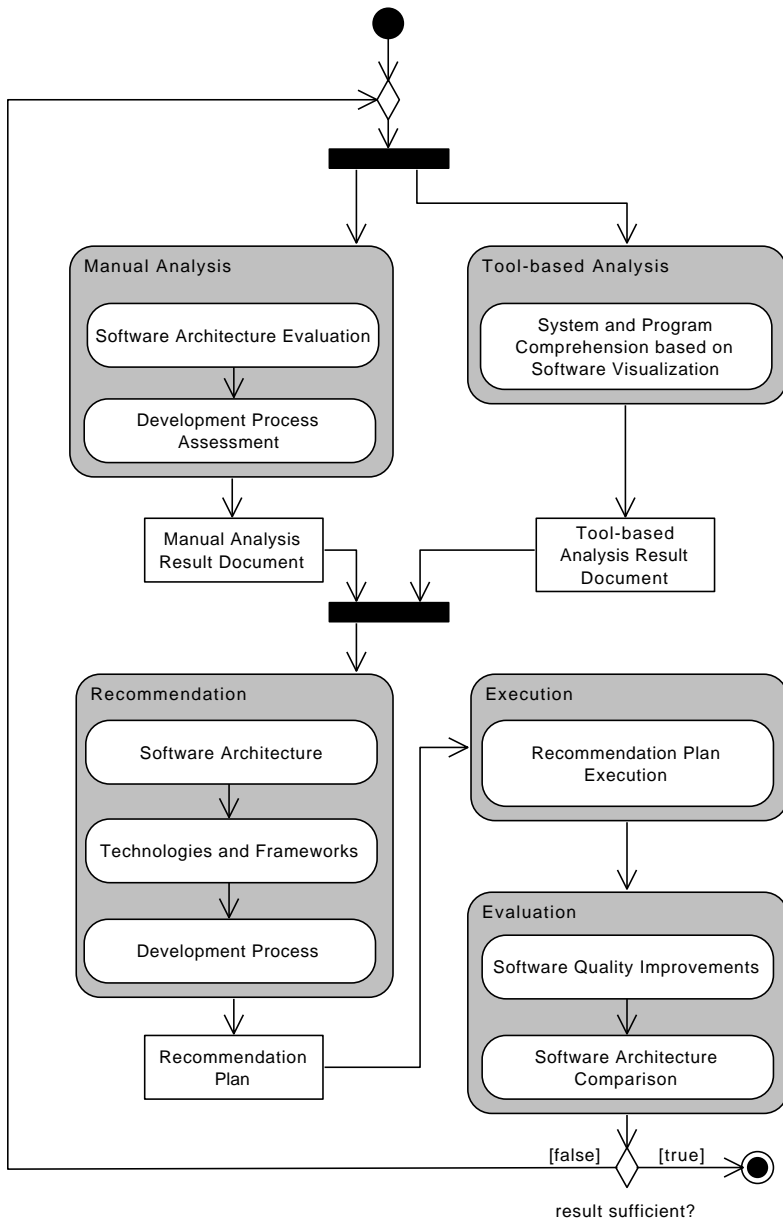


Figure 6.1. UML activity diagram illustrating the CORAL method

## 6. The CORAL Method

### 6.1 Manual Analysis

An existing software and involved systems, which are in need of modularization and modernization, have to be analyzed first (by the developers). Therefore, we need to take a look at the underlying architecture, employed technologies, and tools. This task includes a software architecture and modernization evaluation, in order to identify and reassess legacy source code, frameworks and utilized libraries, and execution environments. The software architecture evaluation task is divided into four parts – (i) a software architecture review, (ii) the application of a software architecture evaluation method, (iii) the identification of technical debt, and (iv) the examination of employed technologies and frameworks. For guidelines and approaches for evaluating software architectures, we refer to [Koz11; KN14; KN16]. Additionally, the developers need to contribute their knowledge of known technical debt, existing documentation, and their current development process. For assessing and evaluating software development processes, we refer to [DN10; CO11]. The results of this activity are summarized in form of a result document.

In the following, we present a more detailed overview of the involved tasks. First, our approach to evaluate the software architecture is described in Section 6.1.1. Then, our approach to assess and evaluate the employed development process is illustrated in Section 6.1.2. Finally, we document and summarize our findings in form of a result document.

#### 6.1.1 Software Architecture Evaluation

In cooperation with the software project developers, we start by reviewing the underlying architecture of the projects' software systems in Section 6.1.1.1. Afterwards, we evaluate the employed software architecture based on a well-established method from literature in Section 6.1.1.2. Likewise an important task within the evaluation embodies the knowledge of technical debt and existing project and source code documentation. Thus, we retrieve the knowledge of existing technical debt and project and source code documentation from the developers in Section 6.1.1.3. Finally, we need

to identify employed technologies and tools utilized in a software project and take a look at the current state-of-the-art and trends in Section 6.1.1.4.

### 6.1.1.1 Software Architecture Review

The underlying software architecture has a huge impact on developed software systems, since it constitutes the foundation or structure of a software system [Has18]. It massively affects how much effort is necessary to e.g., evolve, extend, maintain and deploy a software system, especially in large software landscapes with a high number of systems and applications. Based on this knowledge, we need to gather information about the utilized software architecture from the developers. We focus on the following aspects:

- ▷ Is the employed software architecture (still) suitable for the project or software system?
- ▷ Is the underlying software architecture state-of-the-art or should it be replaced in order to face existing issues?
- ▷ Does the employed software architecture support maintenance and extensibility in a sufficient way?

For guidelines and approaches for evaluating software architectures, we refer to [CKK+03; BT06; BG09; Koz11; KN14].

### 6.1.1.2 Software Architecture Evaluation Method

In addition to the performed software architecture review, we conduct a systematic software architecture evaluation based on a well-established method from the literature. In the past, several literature surveys regarding software architecture evaluation methods have been conducted in research [RG08; PS15] and industrial contexts [BN12]. Based on these reports and the requirements within our process, we chose the *Architecture Tradeoff Analysis Method* (ATAM) [KKC00] as our designated software architecture evaluation method. It is a well-established method and has been utilized to evaluate several types of software architectures, also in the industrial context, over

## 6. The CORAL Method

the past two decades. Furthermore, it has also been used for migrating software systems [KN14; CAH+19]. For case studies, guidelines, and best practices employing ATAM, we refer to [FHL02; BWS+06; GSI15; CPD+16; Sta17b; PDD+17; CAH+19].

### 6.1.1.3 Technical Debt Identification

Technical Debt embodies a huge problem in software development. Primarily, it affects the evolvability and maintainability of software systems in a negatively manner. Thus, it is an important task within the evaluation to retrieve knowledge of existing technical debt and project and source code documentation. Therefore, we need to start with discovering existing technical debt. It is also possible to employ tools, like static code analyzers, to identify technical debt. However, employing such tools can involve the danger to equate the findings with the existing technical debt [KNO12]. Therefore, we need to talk to the developers about already known technical debt first. In detail, we focus on three aspects of the technical debt landscape, namely (i) *architectural and structural debt*, (ii) *documentation debt*, and (iii) *low internal quality*. If technical debt exists, we need to assess its impact and related severity on current software systems. For approaches supporting discovering existing technical debt and refactoring strategies, we refer to [Let12; EBO+15; MBC15; LAL15; TMP+19].

### 6.1.1.4 Technology Examination

The examination of technologies and tools utilized in a software project constitutes also a major part in our *Manual Analysis* action. First, we examine employed programming languages, frameworks, utilized libraries, and execution environments in collaboration with the developers. Then, we take a more detailed look into our findings and discuss whether they are still state-of-the-art or if some or all of them could be replaced. We focus on the following aspects:

- ▷ What programming languages are employed? Are functional or technical constraints or dependencies existing?

## 6.1. Manual Analysis

- ▷ Which technologies and frameworks are used?
  - ▷ How do they affect the architecture of the software?
  - ▷ Are they exchangeable with reasonable effort?
  - ▷ Are they still state-of-the-art or at least further maintained?
- ▷ What kind of execution environments are utilized for the development of the software system?
  - ▷ Is special hard- or software needed to deploy the software system?
  - ▷ Can the deployment be automatized in a reasonable manner?

Apart from a large number of practical guides for evaluating technology stacks on the internet, there also exist several approaches in the literature, which could be applied to examine and evaluate employed and available technologies, frameworks, and tools [CXH16; CX16a; CX16b; MLS17].

### 6.1.2 Development Process Assessment

The employed development process always affects a software project and its included software systems. Based on this knowledge, we need to gather information about the development process from the developers. We focus on the following aspects:

- ▷ What development procedure is used? Is it more like a sequential process (e.g, waterfall model) or rather an iterative approach (e.g., Scrum)?
- ▷ Are negative impacts within the development process already known, which reduce the internal or external quality of the software system?
- ▷ Is a configuration management software used? If so, are standard-procedures documented?
- ▷ Is there a building pipeline utilized? What kind of tests are existing and how is the related workflow organized?

## 6. The CORAL Method

- ▷ Is there a transparent information policy existing and accessible for the developers? Examples are (interface) documentation, guidelines, best practices, employed standards, and process descriptions.

For assessing and evaluating software development processes, we refer to [DN10; CO11; AC15; KKN+19].

### 6.2 Tool-based Analysis

Afterwards, the software system is analyzed with tools, which aid the modularization process by detecting (technical) flaws, possible shortcomings, and optimization potential. In detail, we focus on the understanding of software systems. We address this aspect by applying the software visualization tool *ExplorViz* to aid the system and program comprehension process. We employ *ExplorViz* to achieve a better understanding of the software systems we want to modularize and modernize within our approach. *ExplorViz* was already successfully utilized for comprehension purposes in several scientific [HJZ+17; HZJ17] and industrial collaborations. By utilizing *ExplorViz* for the program comprehension process, we take advantage of software visualizations instead of software artifacts like source code or documentation. Thus, we can enhance our previously obtained knowledge about the software systems from discussions and interviews with the software developers. Finally, we document our findings in form of a result document.

### 6.3 Recommendation

In this activity, we take a look into the analysis result documents of the *Manual Analysis* and *Tool-based Analysis* actions, and design a recommendation plan in collaboration with the developers. The recommendation plan is based on the results and examines possible (target) architectures, technologies, and frameworks. Thereby, we also take the employed development process into account. The purpose is to facilitate synergy effects between the software system and the corresponding development process. In the



best case, we achieve a collaborative development process, which supports the planned modularization and modernization from the beginning.

### 6.3.1 Software Architecture

The aspect of software architecture plays an important role while designing a recommendation plan. An established way to modernization, modularization, or migration of software architectures constitute architectural (migration) patterns. In recent years, cloud and microservice migration patterns became mature and received attention within such processes.

One solution depicts a cloud migration for a transition to a different target architecture. Cloud migration approaches range from the decision making processes to facilitate legacy software migrations with approaches based on best practices, experiences, and lessons learned [JPC+15]. Approaches, like the model-based approach CloudMIG Xpress [FH11], allow to perform a migration towards cloud based environments with a scalable and resource-efficient deployment. For an overview of recent strategies and cloud migration solutions we refer to [JAP13; PXW13; ZZ14; JPC+15]. More recently, microservices architectures seem to become the new state-of-the-art for designing and implementing lightweight and easily deployable software systems. Thus, migration approaches and patterns based on this architectural style also state an alternative target architecture. For comparative works about microservices and microservice patterns we refer to [BHJ16; TLP18; BHJ+18].

### 6.3.2 Technologies and Frameworks

Choosing suitable technologies and frameworks within a recommendation plan also depicts an important task. In the case, that there are no restraints regarding utilized technologies and frameworks, we are able to select and evaluate potential candidates for different aspects of the software system. Based on our own experience this process is highly practically-driven and bases on experiences and lessons learned from previous projects and

## 6. The CORAL Method

collaborations. Despite that, there exist several approaches in literature which support the selection process [Mwe14; TXA+15; RMG17; MMA+20].

### 6.3.3 Development Process

After the development process has been analyzed in Section 6.1.2, we aim to improve the actual software development process and thus increase the collaboration and efficiency, since the software development process influences the software quality [SG16]. There exist several approaches, which focus on different aspects of the development process, like the communication between developers and non-developers [CYC14] or improving the software teams by employing gamification elements to perform personality assessments [YYO+16]. Another aspect constitutes the employed software development life cycle (SDLC) process model. The actual employed model might be not the best choice for a project. For comparative works on SDLCs we refer to [Awa05; VB16; SS17]. For an overview of best-practices and avoidable failures, we refer to [AD16; KK16; MCS+17; KKN+19].

## 6.4 Execution

After discussing the presented options leading towards a recommendation plan in the last action, we need to prepare the execution of it. More precisely, we work out a proof-of-concept implementation of the recommendation plan first. Thus, we can verify the necessary technical adaptations in general and are able to perform the reengineering and modularization process afterwards on a solid basis. Refactoring tools can help with a realization of the recommendation plan. A comparison of available tools and a guidance towards refactoring was published recently by Santos et al. [Sdd+18]. For an overview of applicable reengineering approaches and occurring challenges we refer to [RS16; MES+19].

## 6.5 Evaluation

Once we executed our recommendation plan, we need to evaluate its impact on the software system. Therefore, we focus on comparing the software quality based on metrics provided by software quality tools on one hand and the software architecture through visual comparison on the other hand. Typically, the results of the evaluation are not sufficient after only a single execution. Thus, it is likely, that the overall approach needs to be conducted multiple times to achieve an acceptable state.

### 6.5.1 Software Quality Improvements

In order to verify if the software quality has been improved after a performed iteration of the *CORAL* approach, we use tools to support our technical debt analysis process. Thus, we employ the cloud-based open-source platform *SonarCloud*<sup>1</sup> to discover and confirm existing technical debt in the software project. *SonarCloud* and its non-cloud variant *SonarQube*<sup>2</sup> are tools for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on various programming languages. Additionally, they offer reports on several metrics, e.g., duplicated code, unit tests, code coverage, code complexity, bugs, and security vulnerabilities [PC13; MA10]. *SonarCloud* and *SonarQube* are utilized in industry and research alike [MJL+15; JDP19; TJL16].

### 6.5.2 Software Architecture Comparison

Another important aspect within the *Evaluation* action embodies the comparison of the software architecture before and after the performed iteration. For this task, we again employ software visualizations, as proposed by Beck and Diehl [BD13]. More precisely, we employ an extension of *ExplorViz* which allows to visually compare two versions of a software system regarding their architecture [Weg18]. Another option is to define a target

---

<sup>1</sup><http://www.sonarcloud.io>

<sup>2</sup><https://www.sonarqube.org>

## 6. The *CORAL* Method

architecture upfront and compare it with the modularized software after the iteration [Hac18]. Furthermore, tool support is also advised for performing software architecture evaluation meetings as proposed by [ASB+19; BGH06].

# Tool-based Analysis

An important role within our *CORAL* approach constitutes the tool-based analysis action. This action aims to enhance our knowledge of the software project and involved software systems to be able to eventually present a suitable recommendation plan. We utilize software visualization tools to get a better understanding of the observed software system and its underlying architecture in collaboration with the software developers. Consequently, our approach to understand the software system by employing software visualization is described in Section 7.1. Afterwards, we present in Section 7.2 an extended version of this approach by utilizing Virtual Reality (VR) to improve the comprehension process. In Section 7.3, we extend our presented VR approach towards a collaborative, multi-user approach for improved comprehension in teams. Finally, our developed approach regarding the visualization and comprehension of databases is explained in Section 7.4. The results of this analysis step are summarized in the form of a result document.

## 7. Tool-based Analysis

### Previous Publications

Parts of this chapter are already published in the following works:

1. Robert Heinrich, Reiner Jung, Christian Zirkelbach, Wilhelm Hasselbring, and Ralf Reussner. “An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications”. In: *Software Architecture for Big Data and the Cloud*. Edited by Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim. Cambridge: Elsevier, June 2017, pages 69–89
2. Robert Heinrich, Christian Zirkelbach, and Reiner Jung. “Architectural Runtime Modeling and Visualization for Quality-Aware DevOps in Cloud Applications.” In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA 2017)*. 2017
3. Alexander Krause, Christian Zirkelbach, and Wilhelm Hasselbring. “Simplifying Software System Monitoring through Application Discovery with ExplorViz.” In: *Symposium on Software Performance 2018: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*. Nov. 2018
4. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019
5. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
6. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
7. Christian Zirkelbach. “Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems.” In: *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*. Hanover, Germany, 2017

8. Christian Zirkelbach and Wilhelm Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Research Report 1901. Kiel University, Feb. 2019

## 7. Tool-based Analysis

# 7.1 Program Comprehension through Software Visualization

Understanding a software system is a crucial task when conducting reengineering and modularization tasks. In addition to our software architecture evaluation in Section 6.1.1, we further support the program comprehension process by utilizing software visualization. In the area of software engineering, exploring and understanding software systems is often handled through 2D or 3D visualizations [WL07; WWF+13].

### 7.1.1 *ExplorViz*

Within our approach, we employ the software visualization tool *ExplorViz*<sup>1</sup> for this task. *ExplorViz* uses dynamic analysis techniques, i.e. instrumenting software systems, to provide a live trace visualization of large software landscapes [FRH15; Flo16]. It has the objective to aid the process of system and program comprehension for developers and operators and was successfully employed in several collaboration projects [HJZ+17; HZJ17; KZH+20] and controlled experiments [FKH15b; FFH+15]. Thus, it offers an additional point of view in contrast to manual code analyzes and existing project and software documentation.

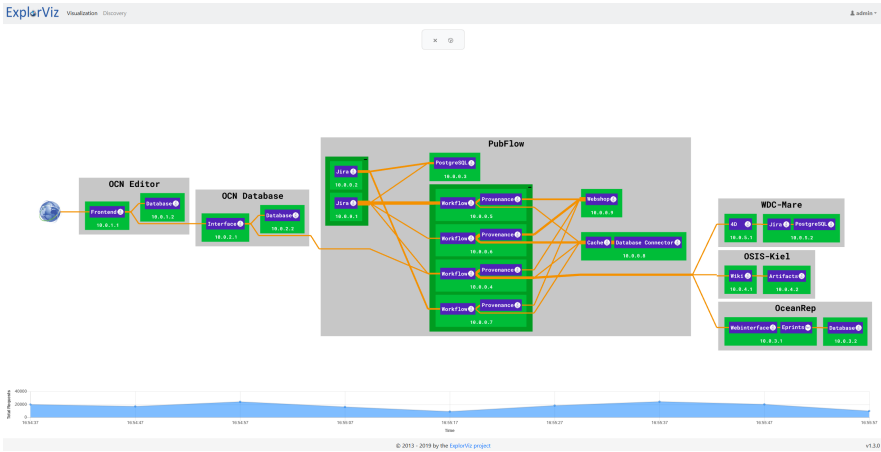
*ExplorViz* features two complementary visualization perspectives – a two-dimensional landscape-level and a three-dimensional application-level perspective. The landscape-level perspective, as shown in Figure 7.1, illustrates a whole software landscape containing software systems, nodes, applications, and communication in-between. A gray box represents a software system, containing nodes (physical or virtual servers) in form of light green boxes. Applications, colored as purple boxes, are running on nodes. Once the same set of applications is running on multiple nodes, they are stacked towards nodegroups (dark green boxes). If applications within the landscape communicate, an orange communication line is drawn between them. The width of a communication line indicates the amount of requests

---

<sup>1</sup><https://www.explorviz.net>



## 7.1. Program Comprehension through Software Visualization

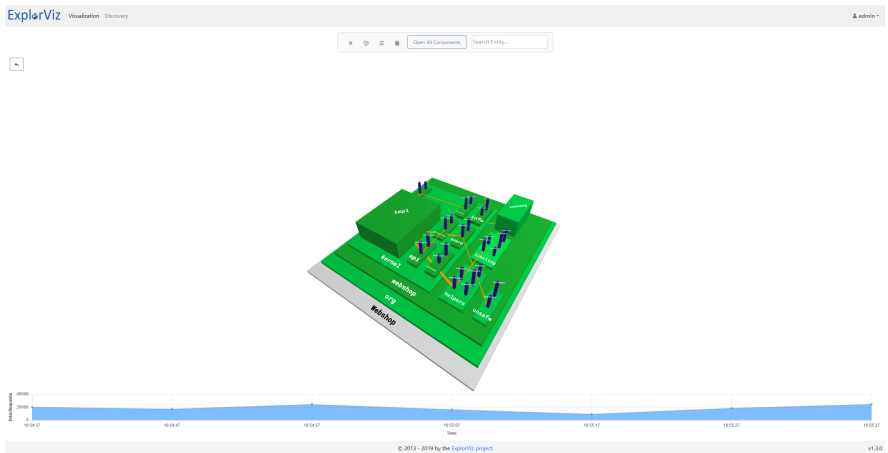


**Figure 7.1.** Landscape-level perspective of *ExplorViz*, Version 1.3.0

between the related applications within a specific time span. For reasons of clarity and comprehensibility, systems, nodegroups, and nodes may be opened and closed in the visualization.

The application-level perspective, illustrated in Figure 7.2, allows the comprehension of a single application within a software landscape. The visualization layout bases upon the city metaphor [WL07]. Here, the green boxes represent packages and thus the structure of the application. Again, for reasons of clarity and comprehensibility, the packages can be opened and closed. Packages can contain classes, which are drawn as purple boxes. The height of a class corresponds to the number of active instances during run time within a specific time span. Communication between classes, e.g., Class A calls a method of Class B, is visualized in the same manner as communication within the landscape-level perspective. The same applies for the width of the communication line respectively number of related requests.

## 7. Tool-based Analysis



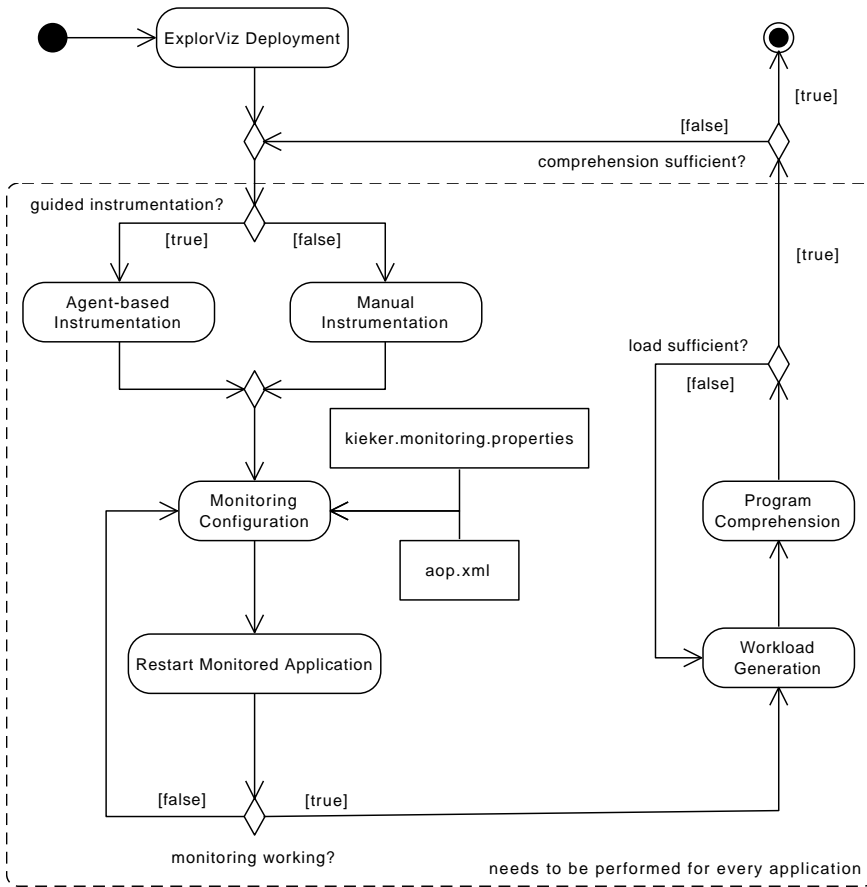
**Figure 7.2.** Application-level perspective of *ExplorViz*, Version 1.3.0

### 7.1.2 Program Comprehension Workflow

We employ *ExplorViz* to achieve a better understanding of the software systems we want to modularize and modernize with our approach. *ExplorViz* was already successfully utilized for comprehension purposes in several scientific [HJZ+17; HZJ17; KZH+20] and industrial collaborations [Eic17; Weg18; Len19].

In order to understand the program comprehension process using *ExplorViz*, we illustrate the workflow in form of a UML activity diagram in Figure 7.3. This workflow is executed in collaboration with the software developers due to required knowledge about the software landscape, systems, and included applications. In the following, we describe the typical workflow for performing system and program comprehension utilizing *ExplorViz*.

## 7.1. Program Comprehension through Software Visualization



**Figure 7.3.** UML activity diagram showing the program comprehension workflow using *ExplorViz*.

### *ExplorViz* Deployment

The first action within the workflow embodies the *ExplorViz Deployment*, which could take place on a developers computer or a dedicated server. *ExplorViz* provides pre-configured artifacts for several use cases by em-

## 7. Tool-based Analysis

ploying *Docker* images, which are hosted on *Docker Hub* [ZKH19b].<sup>2</sup> The corresponding *Docker Compose* files, e.g. release version 1.5.0 or a distributed deployment configuration setup, are available on *Github*.<sup>3</sup> This eases the setup process, because there are no other prerequisites than *Docker* and *Docker Compose*.<sup>4</sup>

### Instrumentation

For instrumentation and monitoring aspects, *ExplorViz* utilizes a dynamic analysis approach provided by the *Kieker* monitoring framework [HRH+09; HWH12; HH20]. In order to instrument a single or multiple applications within a software landscape, the user has to follow a specific procedure for each desired application (represented by the dashed line). Within *ExplorViz* we offer two different options for setting up and configuring applications with *Kieker*. The first option is to perform a *Manual Instrumentation*, which requires a lot of manual effort and consulting documentation regarding the monitoring configuration<sup>5</sup> and consulting the *Kieker User Guide* [Pro17]. Basically, the application-specific configuration embodies two configuration files, namely `aop.xml` and `kieker.monitoring.properties`.

The first configuration file, `aop.xml`, specifies the scope of the monitoring and is used by *Kieker* to perform an automatic source code insertion via Aspect-Oriented Programming (AOP), in the case of Java by AspectJ. More precisely, it defines which parts of the application (defined on the source code level) should be instrumented (`include`) and which ones not (`exclude`). The latter aspect is important, because sometimes the monitoring fails because of employed software like embedded libraries. Additionally, aspects (probes) could be configured, which determine what kind of monitoring is applied to the instrumented source code, e.g., every called operation without mutator methods (`...FullInstrumentationNoGetterAndSetter`) or also database calls executed with JDBC (`...database.FullInstrumentation`).

---

<sup>2</sup><https://hub.docker.com/u/explorviz>

<sup>3</sup><https://github.com/ExplorViz/docker-configuration>

<sup>4</sup><https://www.docker.com/get-started>

<sup>5</sup><https://github.com/ExplorViz/docs/wiki/Monitoring-Configuration>

## 7.1. Program Comprehension through Software Visualization

For more information about monitoring configuration options we refer to the website of the *Kieker* project.<sup>6</sup> An excerpt of an example configuration file<sup>7</sup> illustrates the relevant options in Section 7.1.2. Shortened parts of the configuration file are represented by three dots (...).

The second configuration file, `kieker.monitoring.properties`, defines the configuration of Kieker options. As an example, the monitoring output target can be set, e.g., persisting the output log files within the file system or database, or the output could be directly processed via streams using message protocols like TCP, JMS, JMX, or AMQP.

When using *ExplorViz*, within this configuration file only two configuration options are relevant. An excerpt of an example configuration file illustrates the these options in Section 7.1.2. Shortened parts of the configuration file are again represented by three dots (...).

The first option, `kieker.monitoring.applicationName`, allows to set the name of application, which will appear later in the visualization of *ExplorViz*. The second option, `kieker...SingleSocketTcpWriter.hostname`, defines the ip address or hostname of the host where *ExplorViz* has been deployed. Since the default setting depicts `localhost`, this option only needs to be changed if the monitoring with Kieker is running on a different host or server than *ExplorViz*.

Following this procedure is acceptable for a small number of locally running applications, but is not suitable for enterprise software landscapes with a vast number of systems, server nodes, and applications. Therefore, *ExplorViz* offers an alternative, guided and easy-to-use application monitoring approach, which semi-automatically configures, instruments, and monitors (Java) applications. This agent-based instrumentation allows a user to configure the instrumentation of applications running on a single node via the user interface of *ExplorViz*, after installing an agent application on this node. The architecture and workflow of this alternative application discovery approach in combination with an additional usability evaluation, is described in detail in [KZH18].

---

<sup>6</sup><http://kieker-monitoring.net>

<sup>7</sup><https://github.com/ExplorViz/sampleApplication/blob/master/META-INF/aop.sampleApplication.xml>

## 7. Tool-based Analysis

```
1  <!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.aspectj.org/
    dtd/aspectj_1_5_0.dtd">
2  <aspectj>
3  <weaver options="-verbose">
4  ...
5  <!--
6  Use the exclude/include directives to specify which classes are (not)
    to be considered for weaving.
7  Some examples are given below. Please refer to the AspectJ
    documentation to learn more on the
8  allowed patterns, including wildcards.
9  -->
10
11 <!--
12 Use * to consider all classes in all packages. In this case, you
    typically need some additional excludes.
13 -->
14 <include within="net.explorviz.sampleApplication..*" />
15
16 <!--
17 Important: The logger library that is configured to be used by Kieker
    must always be excluded!
18 (Particularly important when using the include-all directive from
    above.)
19 -->
20 <exclude within="org.apache.commons.logging..*" />
21 <exclude within="org.slf4j..*" />
22 <exclude within="java.util.logging..*" />
23 <exclude within="org.apache.log4j..*" />
24 ...
25 </weaver>
26 <aspects>
27 <!-- Method executions and method calls: -->
28 ...
29 <aspect name="kieker.monitoring.probe.aspectj.flow.operationExecution
    .FullInstrumentationNoGetterAndSetter" />
30 <aspect name="kieker.monitoring.probe.aspectj.database.
    FullInstrumentation" />
31 ...
32 </aspects>
33 </aspectj>
```

**Listing 7.1.** Excerpt of an example aop.xml configuration file of Kieker.

## 7.1. Program Comprehension through Software Visualization

```
1  ...
2  ## The name of application running Kieker. If empty the name will will
   be empty too.
3  kieker.monitoring.applicationName=sampleApplication
4  ...
5  ## The hostname the SingleSocketTcpWriter connects to.
6  kieker.monitoring.writer.tcp.SingleSocketTcpWriter.hostname=localhost
7  ...
```

**Listing 7.2.** Excerpt of an example `kieker.monitoring.properties` configuration file of Kieker.

### Workload Generation

Once the applications within the software landscape have successfully been instrumented, we need to apply (realistic) workload onto them in order to perform a comprehension process with *ExplorViz*. For an overview of workload and measurement options within several application domains, we refer to a comprehensive survey presenting the state of the art of workload characterization [CMT16]. In the past, we already conducted experiments involving program comprehension tasks using *ExplorViz* with generated workload in several collaboration projects [ZA16; HJZ+17; HZJ17] and experiments [FH15]. In many cases we employed the tool Apache JMeter<sup>8</sup> for generating workload. Apache JMeter can be utilized to test the performance on static and dynamic resources in the context of web dynamic applications. It can be used to simulate heavy load on a server, a group of servers, a network or other objects to test their stability or to analyze overall performance under different types of workload. For more information about the usage of JMeter and optimization strategies for generating workload, we refer to [GM17; PS19; VHS+18].

An excellent exemplary software system within the workload context constitutes the JPetstore based experiment suite created by [JA18]. The suite includes distributed and single service variants with and without instrumentation, proper workload drivers, and experimental setups for case studies and creating experiments. Thus, it represents a good starting point

---

<sup>8</sup><http://jmeter.apache.org>

## 7. Tool-based Analysis

on how workload can be generated for distributed, open source software systems.

### **Program Comprehension**

After the monitoring of applications within the software systems is completed and workload is generated and applied, we can begin our main action within the workflow, the *Program Comprehension*. This is a crucial task within our whole approach, because software developers constantly design, implement, maintain, and re-engineer software systems. More precisely, they have to understand the program itself, which is the most time-consuming and costly activity that software developers perform [Sie16; KWF+19].

By utilizing *ExplorViz* for the program comprehension process, we take advantage of software visualizations instead of software artifacts like source code or documentation. Thus, we can enhance our previously obtained knowledge about the software systems from discussions and interviews with the software developers. For applying the program comprehension process on software systems, we refer to several successfully conducted case studies using *ExplorViz* in the scientific [HJZ+17; HZJ17] and industrial context [Eic17; Weg18; Len19]. If the applied program comprehension process with generated workload was sufficient, we can finish our program comprehension workflow, which was illustrated in Figure 7.3. Otherwise, we need either to improve the workload generation or modify the existing monitoring configuration or even extend it, e.g., by adding additional applications.

## **7.2 Improved Program Comprehension with Virtual Reality**

In the past years, Virtual Reality (VR) techniques emerged at the consumer market. Starting with the Oculus Rift DK1 head-mounted display (HMD), which was available at the end of 2013, the VR devices constituted a major step towards the consumer market. Based on this development, modern



## 7.2. Improved Program Comprehension with Virtual Reality

VR approaches became affordable and available for various research purposes. A similar development can be observed in the field of gesture-based interfaces, when Microsoft released their Kinect sensor in 2010 [Gar13]. A combination of both techniques offers new visualization and interaction capabilities for newly created software, but can also improve reverse engineering of existing software by means of immersive user experience. Based on an in-depth 3D visualization and a more natural interaction, compared to a traditional 2D screen and input devices like mouse and keyboard, the user gets a more immersive experience, which benefits the comprehension process [FKH15a]. VR can offer an advantage in comparison to existing development environments to enable new creative opportunities and potentially results in higher productivity, lower learning curves, and increased user satisfaction [EPP15]. Furthermore, recent research is conducted to improve the software engineering process itself with immersive approaches [SMK+18; SMK+19; MSK+19a].

Thus, we go one step further and propose a VR based approach built upon the employed software visualization described in Section 7.1 to understand the observed software systems in another, more immersive way. Our VR approach builds upon the previously shown 3D visualization, which makes use of hierarchical abstractions in order to show details only on demand [FKH15b; FRH15]. Additionally, we employ tracking abilities of HMDs to enable viewpoint rotations and introduce hand- and controller-based interactions to use intuitive motion for *ExplorViz*'s model manipulation. For our VR approach, we use the HTC Vive (Vive)<sup>9</sup> for displaying the software city and the bundled controllers, respectively the Leap Motion Controller<sup>10</sup> (Leap), for the gesture recognition. We build upon our previous work [FKH15a], where we created a first VR version of *ExplorViz* using the Oculus Rift Development Kit 1 as a HMD and Microsoft Kinect v2 camera sensor for gesture control. The major improvements – compared to our earlier version – affect the visualization in terms of performance and extensibility on the one hand, and on the other hand – and more significant – the gesture control with respect to ease of use and accuracy. Furthermore,

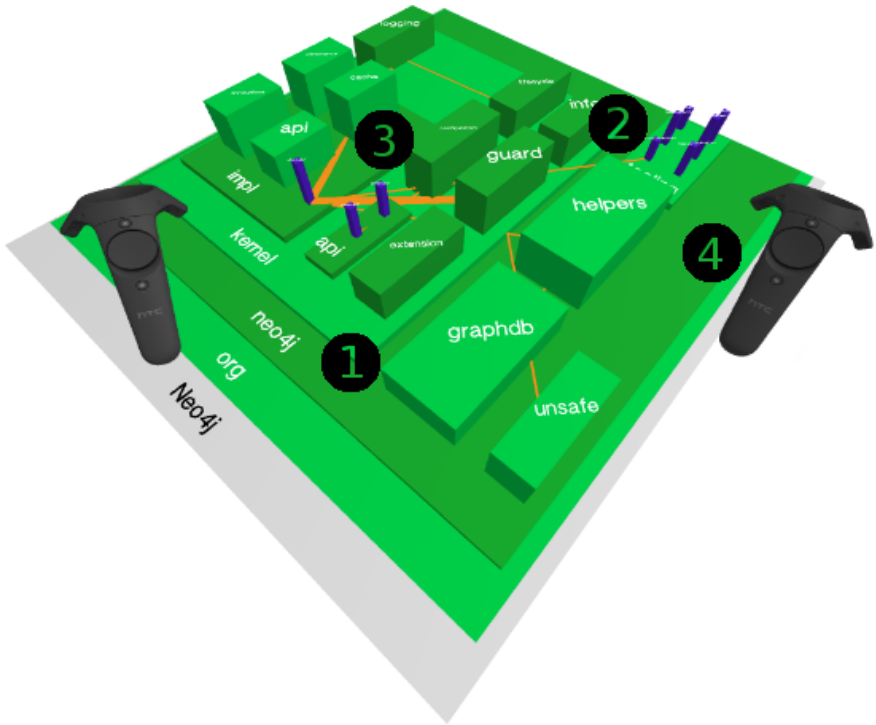
---

<sup>9</sup><https://www.htcvive.com>

<sup>10</sup><https://www.leapmotion.com>

## 7. Tool-based Analysis

the Vive features a perceptible higher display resolution, which provides a better usability, especially for reading labels within our visualization.



**Figure 7.4.** Visualized 3D application-level perspective: Simplified version of the Java application *Neo4j* in VR [ZKH19a].

### Visualization Concept

An example application visualized in the 3D application-level perspective using the VR extension is shown in Figure 7.4. The visualization depicts what a user sees when he wears a HMD and holds controllers within his

## 7.2. Improved Program Comprehension with Virtual Reality

hands. The instrumented Java application is a simplified version of the graph database management system *Neo4j* and is running within *ExplorViz*. The green boxes (❶) in our 3D visualization represent packages showing their contained elements. They can be opened or closed interactively. Classes are visualized by purple boxes (❷) and the communication links are displayed by orange lines (❸). The width of these lines corresponds to the number of executions for a called method or operation during a specific time interval. The height of classes maps to the active instance count of objects for the respective class. For orientation purposes and a better perceived usability, the visualization also contains the Vive controllers (❹), whose position corresponds to their location within the virtual space.

### Technical System

The VR approach employs tracking abilities of HMDs to enable viewpoint rotations and introduces hand- and controller-based interactions to use intuitive motion for *ExplorViz*'s model manipulation. A previous version of our VR approach used an Oculus Rift Development Kit 1 and a Microsoft Kinect v2 sensor. Our new approach uses HTC's Vive HMD as a display and its controllers for interaction, and as an alternative the Leap Motion Controller for hand gesture recognition.

### HTC Vive

The Vive headset has a refresh rate of 90 Hz and a 110 degree field of view. The device uses two OLED panels, one per eye, each having a display resolution of 1080×1200 (which allows a combined resolution of 2160×1200 pixels) and a pixel density of 448 ppi.<sup>11</sup> Inside the headset's outer-shell are dozens of infrared sensors integrated that detect the base stations' IR pulses to determine the headset's current location in a space.<sup>12</sup> Other sensors included are a G-Sensor, gyroscope, and proximity sensor.<sup>13</sup> HTC bundles their HMD

---

<sup>11</sup><http://www.gamespot.com/articles/valve-and-htc-reveal-vive-vr-headset/1100-6425606>

<sup>12</sup><https://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768>

<sup>13</sup><https://www.vive.com/us/product/vive-virtual-reality-system>

## 7. Tool-based Analysis

with two controllers and two laser emitters, called Lighthouse Base Stations. The latter devices are used for inside-out tracking, hence the HMD and the controllers obtain their respective positions in space. The controllers feature multiple input methods including a track pad, grip buttons, and a trigger. Across the top of the controller are 24 infrared sensors integrated, which detect the Vive Lighthouse stations to determine the location of a controller.<sup>14</sup> The SteamVR Tracking system is used to track the controller location to a fraction of a millimeter, with update rates ranging from 250 Hz to 1 kHz.<sup>15</sup> The Lighthouse Base Stations are two (or more) black box-like devices that span a 360 degree virtual space. The base stations emit timed infrared pulses at 60 pulses per second that are then picked up by the headset and controllers with sub-millimeter precision.<sup>16</sup>

### Leap Motion Controller

The Leap Motion Controller instead employs infrared cameras and LEDs to calculate the position of human hands in front of the device. As it is attachable to common HMDs like the HTC Vive, it allows the development of VR applications with hand-based interactions. Using two monochromatic IR cameras and three infrared LEDs, the device observes a roughly hemispherical area, to a distance of about 1 meter. The LEDs generate pattern-less IR light [WBR+13] and the cameras generate almost 200 frames per second of reflected data.<sup>17</sup> This recorded data is then sent to the client computer, where it is analyzed by the Leap Motion software by synthesizing 3D position data by comparing the 2D frames generated by the two cameras. In a 2013 study, the overall average accuracy of the controller was shown to be 0.7 millimeters [WBR+13]. The Leap comes with a Software Development Kit (SDK) for JavaScript and a runtime environment, which needs to be deployed on the client machine.

---

<sup>14</sup><https://www.vrheads.com/exposing-magic-behind-htc-vive-controller>

<sup>15</sup><https://partner.steamgames.com/vrlicensing>

<sup>16</sup><https://www.roadtovr.com/latest-vive-shipping-with-tweaked-base-stations-redesigned-packaging>

<sup>17</sup><https://developer.leapmotion.com/documentation/javascript/api/Leap.Controller.html>

### Client Machine and Communication

To access the Vive and its data in our web-based *ExplorViz* tool, we use the JavaScript API WebVR.<sup>18</sup> The Vive requires a powerful computer to deliver a high and stable frame rate (90 frame per second at least). Our Windows 10 system utilizes an Intel Core i5-6500 processor, an NVIDIA GeForce GTX 1070 graphics card, and 16 GB of RAM. Based on these two different gesture-based devices, we developed custom gestures to interact with our 3D software visualization. This allows the user to interact with our visualization via four gestures *Translation*, *Rotation*, *Selection*, and *Opening / Closing Packages*. In the following, we describe the implementation of these gestures for the Leap Motion Controller for hand gesture recognition on the one hand, and for the HTC Vive controllers as an alternative on the other hand.

### Leap Motion Controller

The hand-based gestures for interacting with *ExplorViz*'s 3D model are partially based on our previous approach as presented in [FKH15a].

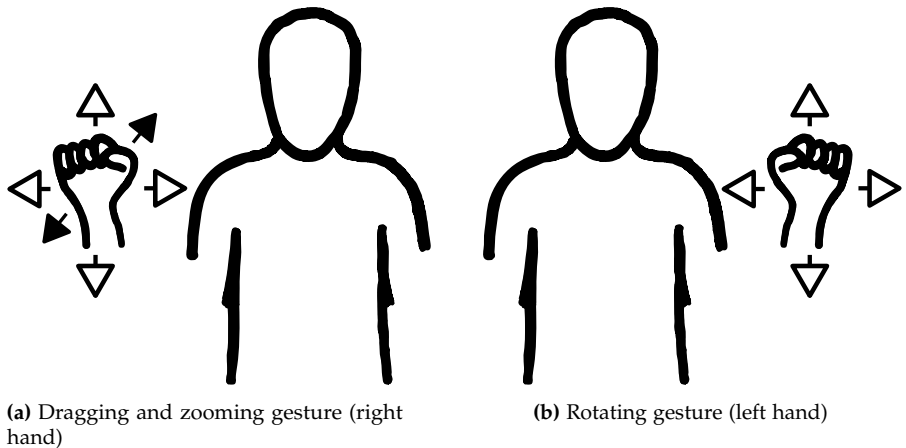
#### Translation and Rotation

Figure 7.5 shows the gestures for moving (dragging and zooming) and rotating the model within the virtual space. If the user wants to move the model, he has to lift his right hand and clench it into a fist first (shown in Figure 7.5a). By moving his hand in a horizontal way, i.e., left and right, he is able to drag the model to the left, respectively right. If the user wants to drag the model vertically (up or down), he needs to move his hand in a vertical manner, i.e, up and down. The user also has the capability to zoom in or out of the model. He just needs to move his still closed fist towards his body (zoom in) or away from his body (zoom out). The corresponding directions are illustrated in form of black arrows within Figure 7.5a. Applying the same gesture to the lifted left hand and also clenching it into fist, allows the user to rotate the model. By moving his

---

<sup>18</sup><https://webvr.info>

## 7. Tool-based Analysis



**Figure 7.5.** VR movement gestures (Leap Motion Controller).

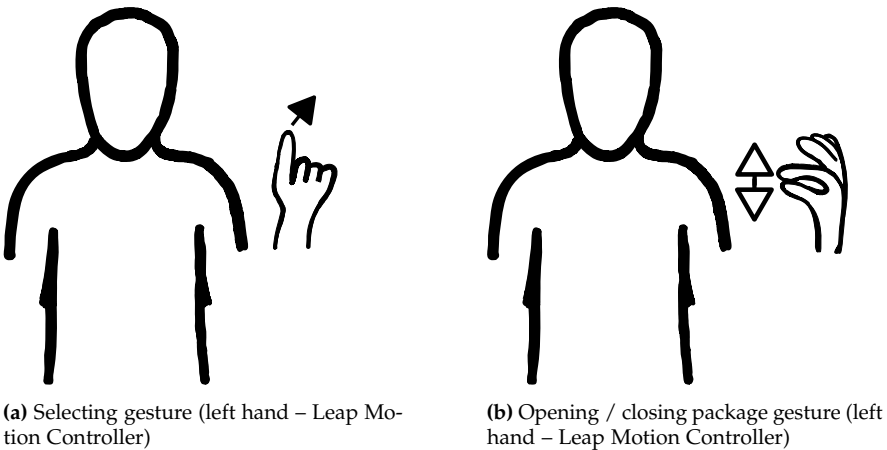
hand in a horizontal way, i.e., left and right, he is able to rotate the model horizontally.

If the user wants to rotate the model vertically, he needs to move his hand in a vertical manner, i.e., up and down (shown in Figure 7.5b). Both gestures are derived from translating real world objects by grabbing and moving them. Additionally, it follows the commonly used dragging and swiping gestures employed on touch screens nowadays.

### **Selection and Opening and Closing of Packages**

Figure 7.6 shows the gestures for interacting (similar to mouse click actions) with the model within the virtual space. To select an entity, which could be a package or class within the visualization, the user needs to lift his right hand and points with the index finger on an entity and pokes onto it (shown in Figure 7.6a). In order to open or close a package, the user again has to lift his right hand first (shown in Figure 7.6b) Then, he needs to press his index finger and thumb together and open them twice (like performing a double click action with a mouse).

## 7.2. Improved Program Comprehension with Virtual Reality



**Figure 7.6.** VR interaction gestures (Leap Motion Controller).

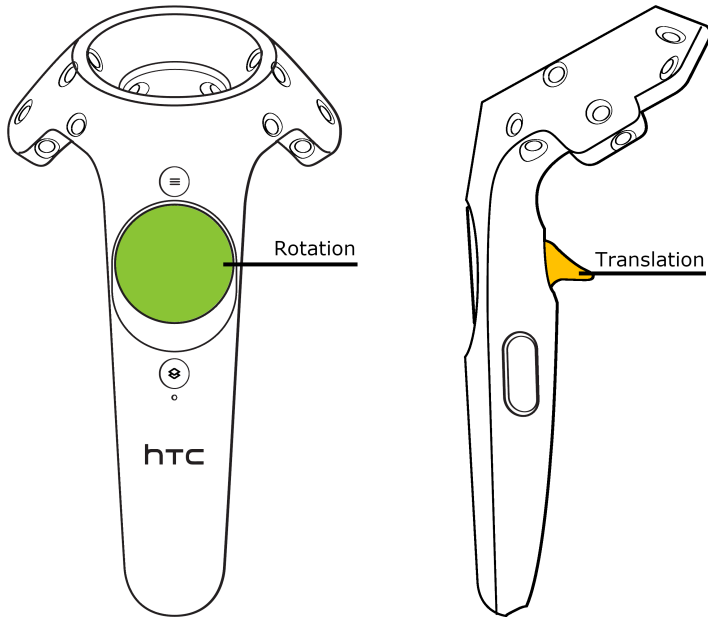
### Controller Interaction

As an alternative towards the previous gesture based interaction, we present a controller-based interaction inspired by modern video game controllers. Thus, we employ the controllers bundled with HTC Vive (actually two, one for the left and one for right hand), which basically constitute improved game controllers with a virtual room-based tracking.

Additionally, due to the position tracking, we are able to use the pointing direction of each controller as a laser pointer, which allows a user to point on an entity within the virtual space. In virtual environments users seem to prefer one-hand interaction, and when two hands are required they prefer interactions that do not require simultaneous hand movements. Instead they prefer interactions which allow them to alternate between their hands [NLL+18]. Consequently, we keep this mind, when designing our interaction concept employing two controllers, one for each hand. The interaction concept, respectively button mapping, is presented in the following.

## 7. Tool-based Analysis

### Translation and Rotation



**Figure 7.7.** VR movement gestures (left controller – HTC Vive).

Figure 7.7 shows the corresponding button mapping (left hand) for moving (dragging and zooming) and rotating the model within the virtual space. The buttons are mapped to the left controller and are used in combination with moving the controller within the virtual space. More precisely, if the user wants to drag the model, he moves the controller in the direction he wants to move the model and presses the translation button (colored in yellow) at the same time. Once he wants the dragging to stop, he releases the button. If the user wants to drag the model vertically (up or down), he needs to move the controller in a vertical manner, i.e. up and down. For dragging the model horizontally, he moves the controller in a horizontal way, i.e., left and right. The user has also the capability to zoom in or out of the model. He just needs to move the controller towards his body (zoom



## 7.2. Improved Program Comprehension with Virtual Reality

in) or away from his body (zoom out) while still pressing the translation button.

### Selection and Opening and Closing of Packages

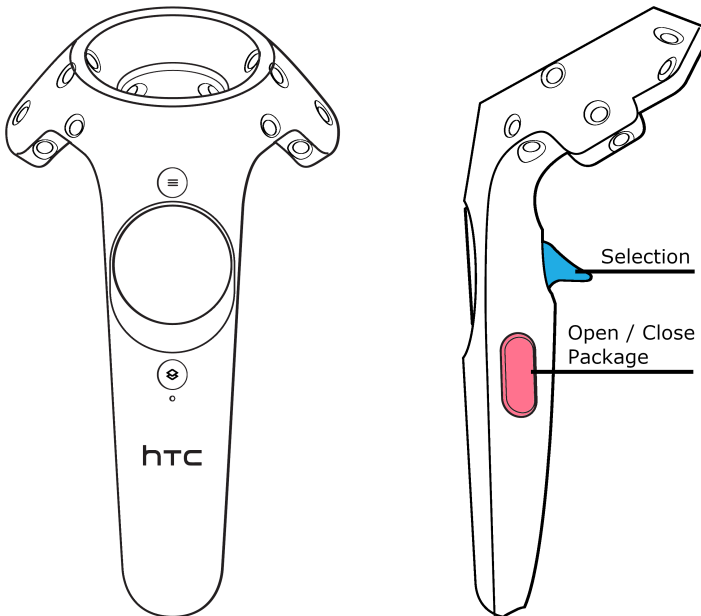


Figure 7.8. VR interaction gestures (right controller – HTC Vive).

Figure 7.8 shows the button mapping (right hand) for interacting with the model within the virtual space. To select an entity, which could be a package or class within the visualization, the user needs to use the laser pointer ability, points on an entity and presses the selection button colored in blue). This action again is derived from performing a double click action with a mouse on the computer. In order to open or close a package, the user needs to perform the same action with a different button, i.e., the *open / close package* button (colored in pink). In comparison to the selection action, the user only needs to press the button twice.

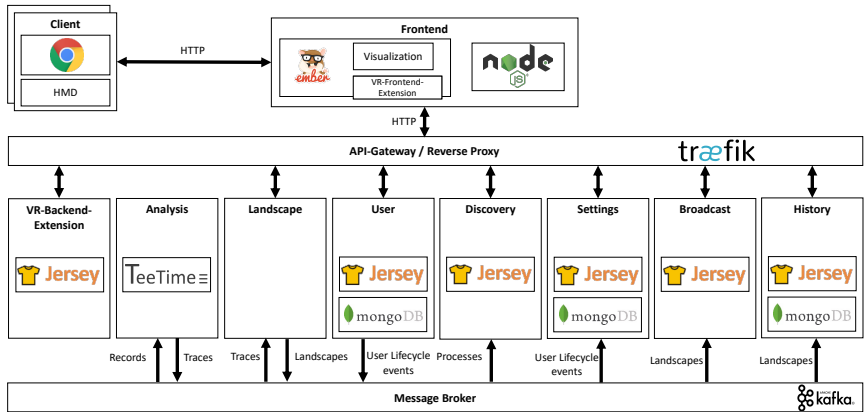
## 7. Tool-based Analysis

### Implementation

In order to evaluate our VR approach in experiments, it has to be implemented first. Therefore, we describe the realized implementation based on the previously presented concept in Section 7.2 focusing on the employed hardware and interaction capabilities of our VR approach. For concrete implementation details related to the source code, we refer to our publicly available GitHub repositories.<sup>19</sup>

For our initial VR approach, we use the HTC Vive (Vive)<sup>20</sup> for displaying the software city and the bundled controllers, respectively the Leap Motion Controller<sup>21</sup> (Leap), for the gesture recognition, as described in Section 7.2.

### Overall architecture



**Figure 7.9.** Architecture of *ExplorViz* in Version 1.5.0 with our realized VR extension showing used software services with employed frameworks and libraries.

<sup>19</sup> <https://www.github.com/ExplorViz>

<sup>20</sup> <http://www.htcvive.com>

<sup>21</sup> <http://www.leapmotion.com>

## 7.2. Improved Program Comprehension with Virtual Reality

Figure 7.9 depicts an overview of the overall architecture of *ExplorViz*, in Version 1.5.0, in combination with our realized VR extension. Based on the provided extension mechanism of *ExplorViz*, we designed and developed a frontend extension for our VR approach. The extension blends seamlessly into the frontend and is accessible via a dedicated menu button called “VR”. First, the HMD and respectively controllers need to be setup, configured, and connected to a client machine. Afterwards, the frontend of *ExplorViz* is accessed via HTTP(S) in a web browser, which also handles the connection between the VR frontend extension and the HMD via the JavaScript API WebVR.<sup>22</sup>

### Leap Motion Controller

Based on these two different gesture-based devices, we developed custom gestures to interact with our 3D software visualization. This allows the user to interact with our visualization via four gestures – *Translation*, *Rotation*, *Selection*, and *Open Packages*, which we explained in detail in Section 7.2.

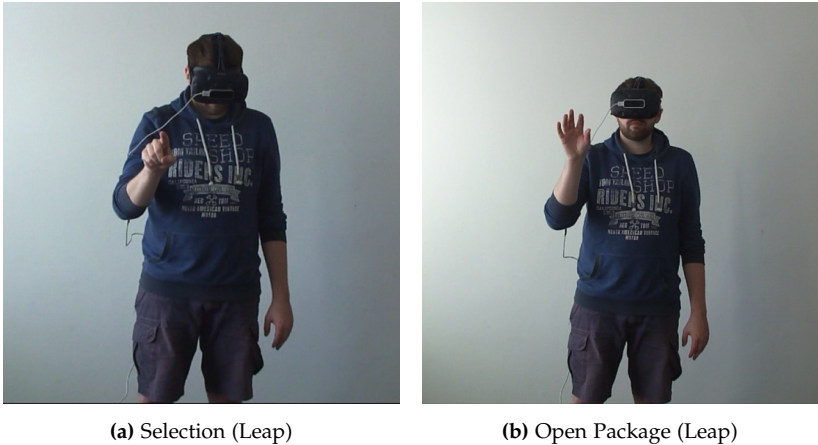
The hand-based gestures for interacting with *ExplorViz VR*’s 3D model are partially based on our previous approach as presented in [FKH15a]. There was, in particular, a lack of an intuitive zooming gesture. We resolved this problem by extending the gesture for translation, which now additionally uses the z-axis for zooming. Now, to move the object freely in space, the user lifts his right hand, clenches into a fist and then starts moving the arm. The same gesture with the left hand is used for rotating the object.

Figure 7.10a demonstrates how packages and classes can be (de)selected for tracking communication between model elements. This discrete gesture is predefined in the Leap Motions JavaScript SDK and can be used out of the box. The user lifts his right hand and pokes the virtual object. This is supported by a crosshair in the middle of the user’s viewpoint. The gesture triggers an event and the object behind the crosshair is (de)selected. In Figure 7.10b, the gesture for opening packages is illustrated. The user lifts

---

<sup>22</sup><https://webvr.info>

## 7. Tool-based Analysis



**Figure 7.10.** VR gestures (Leap Motion Controller).

his right hand and taps the virtual object with a finger. This activates a procedure similar to the selection gesture. Likewise, the user can perform the gesture on an open package to hide its inner structure. This gesture is pre-defined in the Leap Motion's SDK. Additionally, the hands are visualized within the HMD and are thus visible for the user.

Previous tests showed, that common gesticulation in conversations leads to unintended recognition of gestures. The subsequent manipulation of the object often resulted in confusion and annoyance, hence decreasing the usability of a VR mode. To minimize this unwanted behavior we use anchor points.

### HTC Vive Controllers

Although the Leap Motion Controller offers a very natural way of interaction with the system, its false positive rate and angle restriction reduce the usability. For this reason, we present an alternative interaction concept through the HTC Vive controllers, which are also visualized within the

## 7.2. Improved Program Comprehension with Virtual Reality

HMD as previously shown in Figure 7.4 (4). The gesture for *selection* is presented in Figure 7.11a. The user aims for the intended object with a laser-like ray and presses the trackpad. A similar procedure applies for *moving* the 3D model, which is shown in Figure 7.11b. The user lifts the right controller, holds the bottom trigger button, and then moves the object. Further gestures involve *rotating* the model via the trackpad and *opening* and *closing* a package by pressing another button.

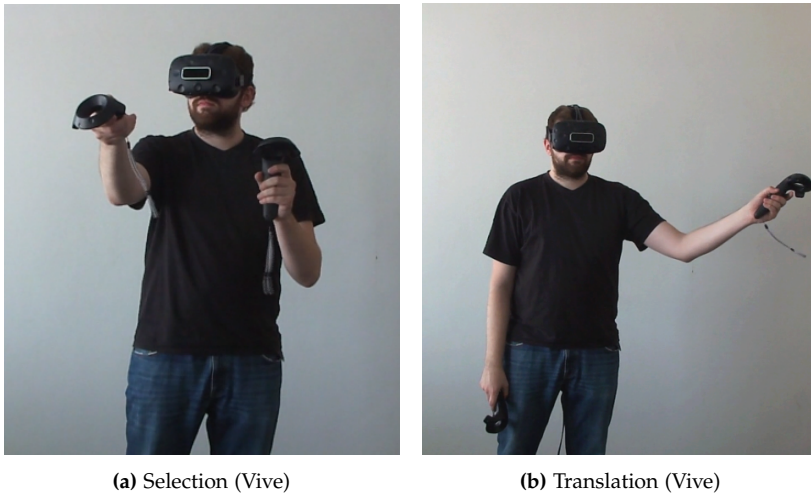


Figure 7.11. VR gestures (HTC Vive).

### Vive Wireless Adapter

In order to verify if the perceived usability was worsened by the wired connection of the HTC Vive, we also tested a wireless adapter kit in addition to our previously described setup. The Vive Wireless Adapter provides a wireless room-scale tracking by substituting the existing cables with a WIFI-based communication between a sender connected to the HMD paired with an external battery pack and a receiver attached to the computer, i.e.,

## 7. Tool-based Analysis

an internal PCIe Intel WIFI card.<sup>23</sup> The Vive Wireless Adapter features a low latency VR experience and allows a user to move more freely within the virtual space without hindrance. Although recent reviews<sup>24</sup> recommended the usage of this extension hardware, we do not endorse this after performing several, extensive tests. One major problem is the wireless setup, which only allows the usage of the adapter in desktop computers, which prevents the usage in mobile setups. The second problem includes occurring WIFI timeouts from time to time, which lead to a poor tracking performance and thus to an unpleasant perceived usability for the user. Last but not least we describe the biggest disadvantage of the adapter – the decreased rendering quality. In an overwhelming majority of our tests we recognized a decreased rendering quality, which is expressed by a blurry visualization and interfering artifacts. This was an unpredictable major drawback of the adapter, which led to the situation that we switched back to the default, wired setup accepting its disadvantages.

### **The VR Approach Shown in a Nutshell**

In order to demonstrate the described gestures, their handling, and related visualizations we uploaded a demo video on Youtube.<sup>25</sup> We start with a short introduction of the visualization and the related elements within our sample system, i.e., packages, classes, and communication. Afterwards, we focus on the interaction and present the gestures, which constitute the controls for the user. As we offer two different gesture recognition systems, we begin with the Leap Motion Controller. We perform the described gestures and present the intuitive interaction. Finally, we use the Vive controllers, illustrate the advantage over the Leap in respect of the false positive rate and angle restriction, and highlight the high accuracy. Additionally, we emphasize the benefits and drawbacks of both gesture-based interaction concepts. Especially the interaction of our VR approach and the differences between both employed gesture-based devices are shown.

---

<sup>23</sup><https://www.vive.com/eu/wireless-adapter/>

<sup>24</sup><https://www.ign.com/articles/2018/11/14/htc-vive-wireless-adapter-review>

<sup>25</sup><https://youtu.be/V5vLuFn1dcQ>

## 7.3 Distributed Multi-User VR Environments

Our previously presented VR approach in Section 7.2 allows software developers and operators in a more immersive way than using the screen and keyboard to explore and understand (their) software systems. Use cases or applicable tasks span from system and program comprehension over architecture evaluation and comparison to performance bottleneck identifications. A drawback of this approach is, that only a single software developer or operator can use our approach at the same time using a single technical setup. Since in most cases these tasks are usually performed in teams of two or more members, the approach is not applicable in an efficient manner.

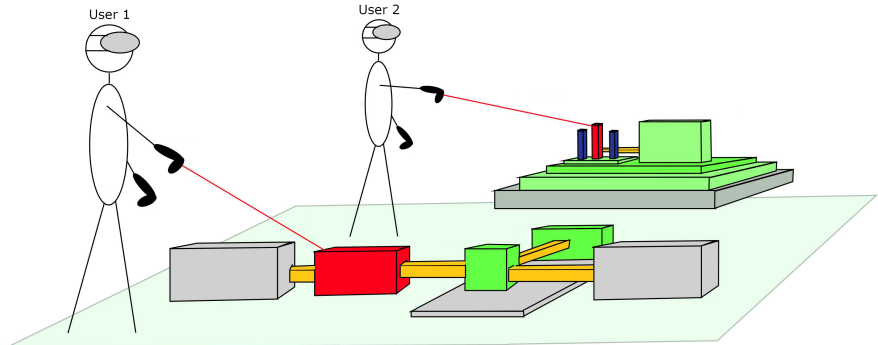
According to Ebert et al. [EKP16] collaboration and teams, processes and organization, sourcing and supplier management, and success factors are the topics gaining the most interest of researchers and practitioners at the moment. Hence, we decided to extend our VR approach such that multiple team members can use it collaboratively at the same time sharing the same virtual environment and thus the visualized software systems. This allows teams to perform their tasks together, even if they are physically distributed over the globe, and we expect that this improves the perceived usefulness and increases the outcome. Thus, our approach is also applicable for globally distributed teams. Since software is nowadays developed by globally distributed teams, projects, and companies in many cases, this states an valuable aspect of our approach.

### Multi-User Concept

The basic idea of the extended VR approach towards a collaboratively usable, distributed virtual multi-user environment is shown in Figure 7.12. In order to allow users efficiently working together with the visualization, it is necessary to merge both existing visualization perspectives, so that the landscape-level perspective is also available in the VR environment and not just the application-level perspective of a single application. Within Figure 7.12, we can see a sketch blending the physical world involving two

## 7. Tool-based Analysis

users (User 1 and User 2), who wear HMDs and hold two VR controllers, one in each hand, with the virtual world showing the software visualization.



**Figure 7.12.** Sketch showing two users employing the Multi-User VR approach collaboratively based on [Häs17].

Each controller allows the users to interact with the visualization, i.e., *Translation, Rotation, Selection, and Opening / Closing Packages*. More precisely, each controller has a virtual laser pointer like ray (colored in red), which allows to interact with objects (see Section 7.2) within the visualization. While exploring a software landscape (which is laying on the virtual floor colored in light green), users can highlight objects like systems, nodes, or communication (User 1 highlights a system in Figure 7.12). In order to differentiate between the rays and highlighted objects within the virtual environment each user has a distinct color. We chose to provide an underlying virtual floor because our supported HMDs are able to operate in a room-scale mode, which means they can move physically within their play area and so also move virtually in our VR approach. This also allows users to go down on their knees or move their head to get closer to the visualization, if necessary. Furthermore, they are able to open included applications, which are afterwards visualized in front of them (User 2 interacts with an opened application in Figure 7.12). Within an opened application, it is possible to interact with its visualized objects like packages, classes, and communication (User 2 highlights a class in Figure 7.12). Additionally, the



### 7.3. Distributed Multi-User VR Environments

users are also represented within the virtual world by rendering their HMD and respectively controllers. Thus, the users are able to see each other in a similar way as in meeting in a real environment. Furthermore, the users are able to communicate with each other by using the integrated headsets within their HMDs in combination with additional software like *Discord*,<sup>26</sup> an instant messaging and VoIP application platform.

## Technical System

In order to be compatible with multiple available HMDs on the market and avoid occurring interaction problems regarding the Leap Motion Controller Sensor in our VR approach, we decided to discontinue the gesture-based interaction capabilities and focus on controller-based interaction. Without an up-to-date HMD the approach is not efficiently usable, so we need to support multiple devices, which also support several hardware and operating system setups on one hand, and are sustainable for upcoming HMDs. Thus, we support four consumer-oriented HMDs, which are previously introduced and available on the market. The HMDs are the previously presented Vive, one of its successors, the Vive Pro, the Oculus Rift, and one of its successors, the Oculus Rift S. The basic technical setup stays the same, so only the newly added HMDs and setup differences are explained in the following.

### HTC Vive Pro

As the Vive Pro headset is a successor of the Vive, most of the specifications stay the same. In comparison, the device now uses two 3,5" AMOLED panels, one per eye, each having a display resolution of 1440×1600 (which allows a combined resolution of 2880×1600 pixels) and a pixel density of 615 ppi.<sup>27</sup> Based on the higher resolution, the Vive Pro offers a 37% ppi (Points-Per-Inch) and 78% Pixel increase compared to the Vive. Furthermore, the HMD has been improved in aspects of comfort providing a 24% larger Face Gasket Surface area for lowered stress. Again, HTC bundles their

---

<sup>26</sup><https://discord.com>

<sup>27</sup><https://www.vive.com/eu/product/vive-pro/>

## 7. Tool-based Analysis

HMD with two controllers, which stayed the same, and two improved laser emitters, named SteamVR Base Stations 2.0. The SteamVR Base Stations 2.0 are, like their predecessors, two (or more) black-box-like devices that span a 360 degree virtual space, which allow a play-area up to 5m×5m. If additional stations are added the play-area can be extended up to 10m×10m.

### Oculus Rift

The Oculus Rift (Rift) headset has a refresh rate of 90 Hz and a 110 degree field of view. The device uses two Pentile AMOLED panels, one per eye, each having a display resolution of 1080×1200 (which allows a combined resolution of 2160×1200 pixels) and a pixel density of 456 ppi.<sup>28</sup>

Similar to the Vive, the Rift's headset has infrared sensors integrated that detect the sensors' IR pulses to determine the headset's current location in a room.<sup>29</sup> Other sensors included are a G-Sensor, gyroscope, and accelerometer sensor.<sup>30</sup> Oculus bundles their HMD with two controllers and two IR tracking sensors, called Oculus Sensors. The latter devices are used for 180 degree front-facing tracking, hence the HMD and the controllers obtain their respective positions in space. A full 360 degree tracking can be achieved by installing a third Oculus Sensor, which needs to be procured separately. The controllers, named Oculus Touch, feature multiple input methods like an analog stick, three buttons, and two triggers. Additionally, integrated capacitive sensors are able to detect finger gestures a user makes while holding them.<sup>31</sup>

### Oculus Rift S

The Oculus Rift S (Rift S) is an improved version of the Rift regarding some specifications, tracking, and bundled controllers. The Rift S headset

---

<sup>28</sup><https://venturebeat.com/2019/03/20/oculus-rift-s-versus-oculus-rift-the-spec-comparison-chart/>

<sup>29</sup><https://uploadvr.com/hackable-webcam-oculus-sensor-be-aware/>

<sup>30</sup><https://venturebeat.com/2019/03/20/oculus-rift-s-versus-oculus-rift-the-spec-comparison-chart/>

<sup>31</sup><https://www.pocket-lint.com/ar-vr/reviews/oculus-rift/136621-oculus-rift-review-vr-virtual-reality-specs>

### 7.3. Distributed Multi-User VR Environments

has a refresh rate of 80 Hz and a approximately 115 degree field of view. Furthermore, the headset now contains two LCD panels, one per eye, each having a resolution of 1280×1440 (which allows a combined resolution of 2560×1440 pixels) with a pixel density of nearly 600 ppi. In this aspect, the Rift S is comparable with the Vive Pro. Another difference are the improved Oculus Touch controllers, which are differently shaped, but offer similar features. The major difference between it's predecessor and the Vive (Pro) is the integrated inside-out tracking, which needs to external sensors for position tracking anymore. This fact allows to use a full 360 degree usage of the Rift S without the need to install sensors or base stations upfront, which makes it easy to carry it along for external uses or similar and thus embodies a huge benefit. Although the Oculus Rift S offers several advantages, the Vive Pro performs better than the Oculus Rift S with regards to comfort, display quality, and compatibility with glasses [MFT+19].

#### Client Machine and Communication

To access the HMDs and its data in our web-based *ExplorViz* tool, we continue to use the JavaScript API WebVR.<sup>32</sup> All employed HMDs still require a powerful computer to deliver a high and stable frame rate (90 frame per second at least). Furthermore, to support all HMDs, we need a system which meets at least the minimum hardware requirements specified by the manufacturers. Thus, we utilize a Windows 10 Professional system installed on an Intel Core i5-6500 processor, a NVIDIA GeForce GTX 1070 graphics card, and 16 GB of RAM.

To be able to use our VR mode collaboratively with several users at the same time, it is important that necessary information is shared among them during a session. More precisely, this includes the position data of a user, i.e., HMD and controllers, including the orientation of the software visualization and involved objects. Thus, in addition to our previously presented VR approach in Section 7.2, we need to provide a centralized server (a separate backend extension for *ExplorViz*), which keeps track of all occurring position and visualization changes and notifies all connected users in order to update their visualization. For performance reasons, we

---

<sup>32</sup><https://webvr.info>

## 7. Tool-based Analysis

decided to base our communication on the established protocol WebSocket, which offers a good performance in web real-time communication [LS12]. The protocol provides a full-duplex communication, while only needing to establish a single TCP connection for every connected user.

### Controller Interaction

Based on the presented four different HMDs, which we want to support within our approach, we further developed our previously presented controller gestures and adapted them for each bundled controller-based device to interact with our 3D software visualization.

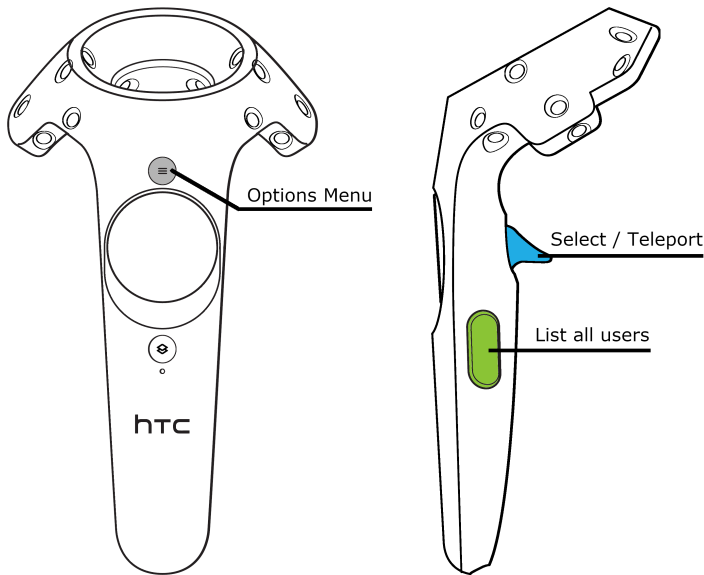
#### HTC Vive and HTC Vive Pro

Within our collaborative and extended VR approach, the user can interact with our visualization via six controller gestures *Select / Teleport*, *List all users*, *Options Menu*, *Opening / Closing Packages*, *Move Application*, and *Display Information*. In comparison to our previous VR approach, we combined the *Movement* and *Rotation* gestures into a single gesture *Move Application* for a better usability. Furthermore, we added three additional gestures (*List all users*, *Options Menu*, and *Display Information*), which enhance the usability of our approach by providing more interaction capabilities, information, and especially multi-user related options, e.g., connecting to the backend extension. In the following, we present the above mentioned controller gestures and use the HTC Vive (Pro) as a reference controller to explain them. The related mapping for the Oculus Rift respectively Oculus Rift S is explained afterwards.

#### User Movement, Selecting, and Options

Figure 7.13 shows the corresponding button mapping (left hand) for moving the user within the virtual space, selecting entities, and multi-user related options. More precisely, if the user wants to move within the virtual space, he has basically two options. Since we now use HMDs, which are capable of room-scale VR, the user is able to move physically within his play-area and

### 7.3. Distributed Multi-User VR Environments



**Figure 7.13.** VR interaction gestures (left controller – HTC Vive (Pro)).

moves alike in the virtual space. The second option allows a user to teleport within the virtual space so that he can move from one edge of the visualized software landscape to the other edge in an easy way. In order to teleport, the user has to hold the teleport button (colored in blue) first. Afterwards, a circle appears on the virtual ground, which sets the teleportation target and is movable with the controller along the virtual floor. Once the user releases the button, he gets teleported to the last position of the circle.

Within the visualization, the user has also the capability to select entities. To select an entity, which could be a package or class within the visualization, the user needs to use the laser pointer ability, points on an entity and presses the selection button (colored in blue). This action again is derived from performing a double click action with a mouse on the computer.

Other capabilities are available by pressing the *Options Menu* button (colored in gray). The *Options Menu* provides, as the name suggests, a menu for advanced options regarding the multi-user mode. One central option

## 7. Tool-based Analysis

within the menu is to establish a connection with the server and thus switch from an offline towards an online operation mode for collaboration with other users. The last button *List all users* (colored in green) allows a user, if he is connected to a server, to get a list of all other currently connected users. The respective button mapping for the Oculus Rift is shown in Figure 7.14

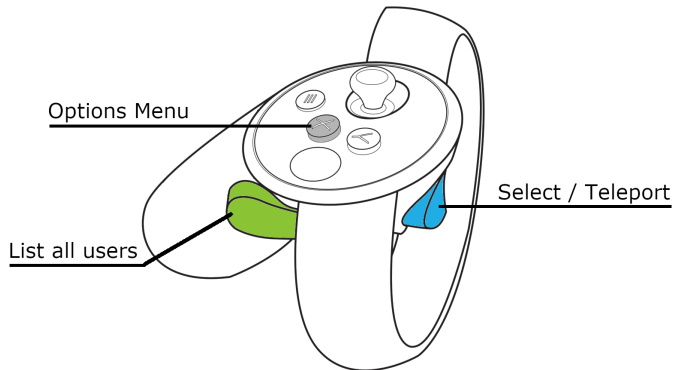
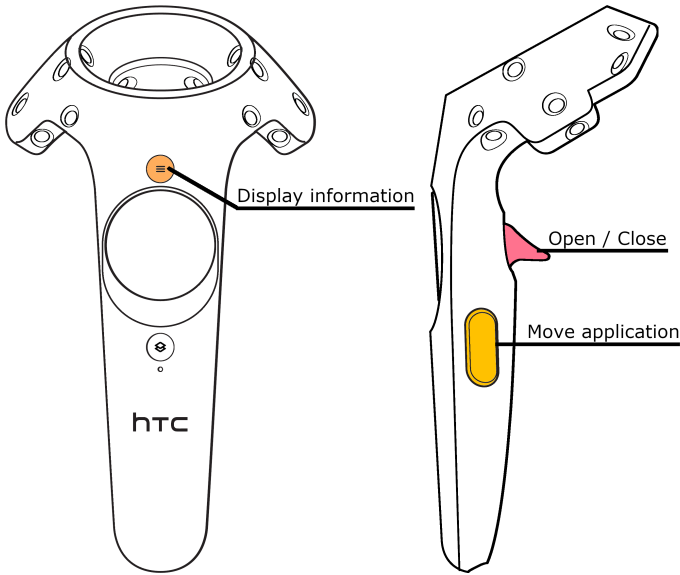


Figure 7.14. VR interaction gestures (left controller – Oculus Touch).

### Visualization Interaction and Displaying Information

Figure 7.15 shows the button mapping (right hand) for interacting with the model within the virtual space. To open or close a package within the visualization, the user uses the laser pointer ability, points on an entity and presses the *Open / Close* button (colored in pink). In comparison to the selection action, the user only needs to press the button twice. This action again is derived from performing a double click action with a mouse on the computer. Furthermore, the user is able to move the visualized 3D software application within the software landscape, as shown in in Figure 7.4, by pressing the *Move application* button (colored in yellow), while again pointing with the laser pointer ability on the 3D software application. Using this gesture, he can drag the application within the virtual space by moving the controller and releasing the button once the desired position has been reached. This also includes rotating and zooming in respectively out, which

### 7.3. Distributed Multi-User VR Environments



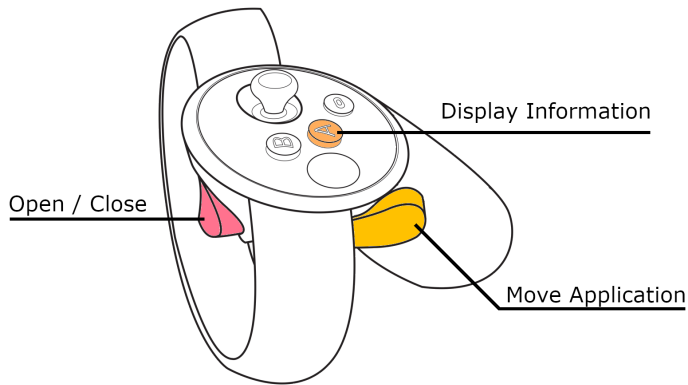
**Figure 7.15.** VR interaction gestures (right controller – HTC Vive (Pro)).

were additional gestures and thus buttons in our previous VR approach. The applied interaction is immediately realized by the rendering within the visualization and thus visible for all connected users. In combination with the visualization of each user, represented by their HMD and respectively controllers, the users are able to collaboratively explore and discuss the visualized 3D software application.

Last but not least, we want to provide additional, useful information to the user compared to our previous approach. For this purpose, we added another functionality to our approach with the button *Display information* (colored in orange). With this gesture, the user is capable to view additional supporting information about entities, which are pointed at by the user. Again, the user needs to use the available laser pointer ability, points on an entity and presses the button. Afterwards, a dialog appears in front of the

## 7. Tool-based Analysis

user, which shows additional information about this entity, i.e, the name, number of contained subpackages and classes of a package, the name of a class and the number of active instances, and the source and target class of a communication, the direction of the communication, and its number of requests and average response time. These information is provided within the (standard) visualization of *ExplorViz* and greatly enhances the system and program comprehension process. For this reason, we integrated them in our improved, collaborative VR approach. The respective button mapping for the Oculus Rift is shown in Figure 7.16.



**Figure 7.16.** VR interaction gestures (right controller – Oculus Touch)

## 7.4 Database Visualization

As mentioned before, large software landscapes often consist of a large number of systems, applications, and communication links. Usually these systems employ databases, which provide well-defined interfaces for retrieving, storing, and processing application data. Databases are pivotal components in large software landscapes and are affected by the growing complexity and evolution of software systems. In combination with the steadily increasing amount of data, it is difficult to maintain a live overview



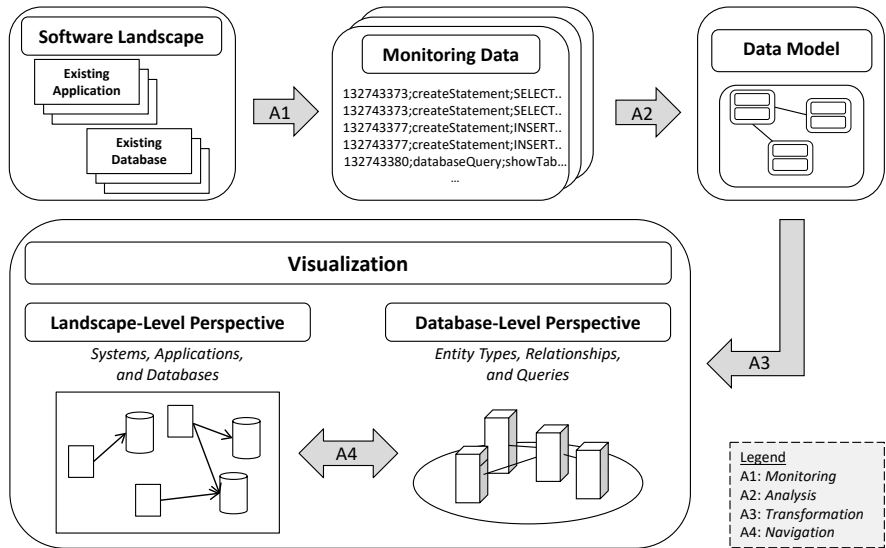
## 7.4. Database Visualization

of these software landscapes, database-related communication, and connected databases. This often leads to insufficient knowledge of the actual structure and behavior of related databases. Additionally, databases are often involved in performance issues within software systems [ZHC15]. Therefore, it is necessary to (i) monitor database queries invoked by applications, (ii) analyze connected databases, and (iii) present the gathered information in an appropriate visualization.

Software landscapes are often visualized with UML-based deployment diagrams [OMG15] or modified versions to keep an overview of the landscape and included application systems [Flo16]. Databases by contrast are commonly represented by entity relationship diagrams [Che76]. To the best of our knowledge, there exists no visualization approach, which offers a software landscape facilitating the communication between applications and databases on the one hand, and the internal structure and actual usage of databases on the other hand.

In [Zir17] we describe problems caused by missing or inappropriate database monitoring and visualization and presented a first sketch of a solution approach. In [ZH19] we propose an improved live visualization approach, which employs a combination of two different, complementary representations, to support the comprehension process of databases and related communication for large software landscapes. Our visualization offers two different views – a landscape-level and a database-level perspective. This combination of visualizations enables operators to understand their databases in detail and analyze database queries invoked by applications at runtime. Based on established visualization concepts like the entity relationship diagrams and the software city metaphor [WL07], operators can be supported in the task of database comprehension. We apply established analysis techniques, namely dynamic and static analysis, to capture necessary information from applications and databases. Possible scenarios of our approach are exploration and enhanced software landscape and database comprehension.

## 7. Tool-based Analysis



**Figure 7.17.** Overview of our *RACCOON* approach for live database behavior visualization.

### The *RACCOON* Approach

Our live database behavior visualization approach *RACCOON* includes four consecutive activities (A1 to A4), which are briefly described in the following. Figure 7.17 illustrates an overview of the activities in our approach.

**A1 – Monitoring:** Within a software landscape existing applications and related databases are *monitored*. We collect conducted database queries in each application, detect which databases are used, and query these databases for structural information directly via their APIs. The results will be provided in form of a data stream, which contains monitoring logs (applications) and structural information (databases).

**A2 – Analysis:** In order to further process the monitoring information, we need to *analyze* it. Analyzing database queries contains basically two

steps – (i) reconstructing the collected monitoring data into corresponding database queries and (ii) aggregating similar ones. For structural information gathered directly from databases, we take the results from the queried APIs and preprocess them for the next activity. Furthermore, we need to store the processed information. The result of this activity is a persistent data model for the whole software landscape, which contains the monitored applications, databases, and their communication among each other. This enables a live visualization of our reconstructed landscape and databases.

**A3 – Transformation:** To enable a visualization of our database queries and related databases, we need to *transform* the data model into a visualization model. This is a necessary step, as the visualization model should only contain relevant information for the requested perspective.

**A4 – Navigation:** Our live database behavior visualization offers two perspectives. The operator is able to view and navigate between them. Each perspective is different and focuses on distinct use-cases – either getting an overview of the software landscape or detailed information of a specific database. Therefore, each perspective employs a different visualization metaphor. Thus, we provide separated views on the landscape-level and database-level within the software landscape. In the following section, we describe the two perspectives, their interaction capabilities, and the navigation between them in detail.

### Live Visualization of Database Behavior

In this section, we describe our live database behavior visualization approach *RACCOON* for large software landscapes in detail. It consists of two complementary perspectives, namely a landscape-level and a database-level perspective. The first perspective employs a UML-based 2D visualization of the reconstructed software landscape. The landscape includes monitored applications and databases and shows their interactions among each other. The second perspective offers a detailed view of a single database by showing its included tables, columns, and relationships. Additionally, executed database queries are visualized in form of connections between tables. Both


## 7. Tool-based Analysis

perspectives avoid problematic color combinations and employ a color vision impaired color design as described in [JK07]. Thus, we only need one color scheme and support visually impaired operators from the beginning.

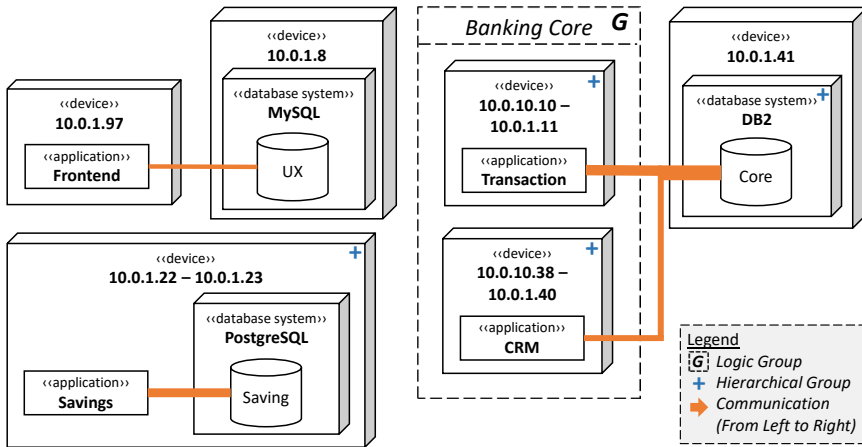
Operators need an appropriate visualization of databases for large software landscapes for comprehension and planning tasks. As databases are affected by evolutionary changes, it is necessary to keep track of changes in the database or queries invoked by applications. This situation requires a live monitoring of relevant applications and databases in order to update the visualizations based on monitoring information. Thus, the analysis rests upon two techniques – (i) dynamic analysis, which implies database communication invoked by applications during runtime, and (ii) static analysis, which embodies the internal structure of a database, namely databases schemes and comprised objects like tables, columns, constraints, and statistical information.

In order to ease the comprehension process for operators, we provide a consistent layout as described in [KLN08]. Changes detected by the monitoring are processed and stored in the data model, transformed into the visualization model, and finally lead towards an updated visualization. In the following, we use the SQL (Structured Query Language) terminology.

### Landscape-Level Perspective

The landscape-level perspective is based upon the UML deployment diagram [OMG15] and provides an overview of the reconstructed software landscape containing monitored applications and databases. Additionally, we employ a data-flow diagram direction-oriented layout [KSS+12], i.e., the data flows from the left (source) to the right (target). In order to distinguish between systems, applications, and databases, we employ UML stereotypes (`<<stereotype>>`). A sketch of this visualization is displayed in Figure 7.18. It shows systems (`<<device>>`), included monitored applications (`<<application>>`), and database systems (`<<database system>>` – showing the vendor) with contained databases (cylinder with the database name). Furthermore, we visualize the communication (edge – colored ) between applications and databases. The thickness of an edge represents the amount of commu-

## 7.4. Database Visualization



**Figure 7.18.** Landscape-level visualization: communication between systems, applications, and databases in a software landscape.

nication, i.e., the more database queries are executed, the thicker the edge. Once a new monitored object is observed, a respective visualization object is integrated into the landscape.


Additionally, we support an automatic visual clustering of similar objects like systems, applications, and databases into hierarchical groups (indicated by the symbol +). This allows us to provide a visual abstraction of monitored objects, which occur more than once within the landscape. By clicking on the symbol (+) the operator is able to reveal individuals of the hierarchical group. For example, the *database system* *DB2* is running multiple times within the *device* *10.0.1.41* containing the database *Core*. Moreover, the operator is able to cluster devices, which contain applications and databases (represented by a dashed rectangle with a name and a *G* in the upper right corner), into logical groups. This feature allows him to add further semantics to the visualization for planning or documentation purposes, e.g., re-deployments or migrations. Finally, the detailed view of a specific database (database-level perspective) is provided by clicking on the






## 7. Tool-based Analysis

respective database within the landscape-level visualization.

### Database-Level Perspective


The database-level perspective is based on the software city metaphor [WL07], a well-established representation in the area of software engineering, and the entity relationship diagrams [Che76], which is a commonly used visualization for relational databases. The visualization shows the internal structure and executed queries within a single database. Figure 7.19a sketches the database tables, relationships, and performed queries during runtime in the database *Core (DB 2)*. The software city metaphor basically provides three different kinds of visualized objects (districts, buildings, and streets). In the following, we describe (i) what information is represented by these objects and (ii) how they behave.

**Districts** Each database (schema) contains one root district. This district is displayed as a round layer with a fixed height (colored ) , as shown in Figure 7.19a, and acts as a foundation for included tables and relationships.

**Buildings** Tables within our database (schema) form buildings within the software city metaphor (e.g., the table *Product*). Each table is visualized as a rectangular box (colored ) with a dynamic height. The default metric used for the height is based on the absolute number of rows stored in the table. It is also possible to apply other metrics, e.g., the number of times the table was involved in queries. Additionally, tables can be opened in order to reveal their columns, as shown in Figure 7.19b, and thus become districts. Furthermore, columns within a table are visualized by small rectangular boxes (colored in several colors) with a fixed height. Figure 7.20 illustrates the semantics of the representation. We separate the columns of a table into three groups based on their constraints. The first group includes columns with a primary key (PK – colored ) . The second group contains columns with a foreign key (FK – colored ) . The third group comprises columns that either have a different constraint (e.g., not null – colored ) or no constraint (colored ) . The layout is based on a single-column (PK), multiple-column (others), and single-column (FK) alignment. Further columns are added in

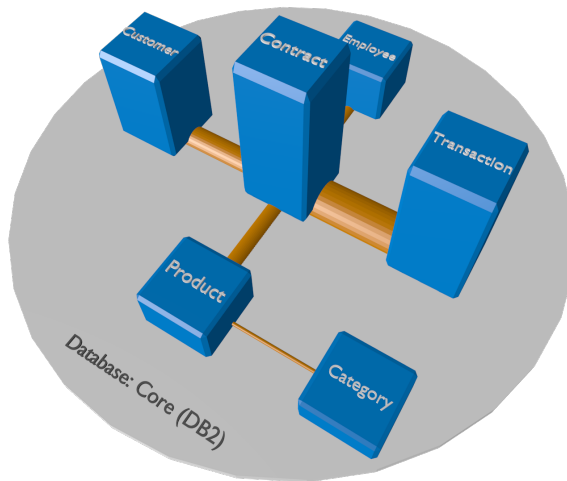
## 7.4. Database Visualization

direction of the dashed arrow (PK and FK: south, others: south-east). In order to ease distinguishing between the groups, we employ separators. Figure 7.21 shows a zoomed-in visualization of the opened table *Product*. It illustrates the contained and colored columns, separated into the three groups mentioned above.

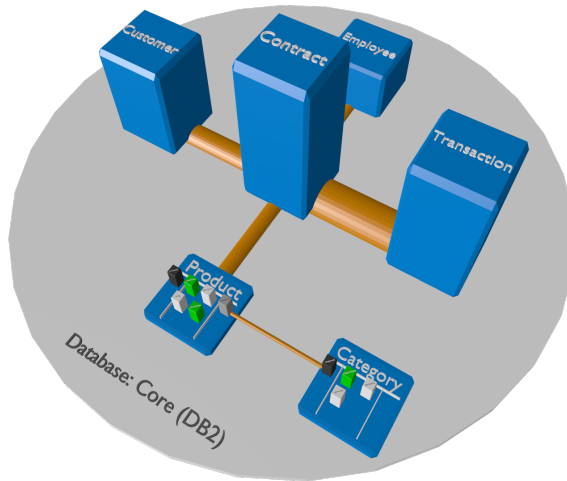
**Streets** Streets represent the communication between tables, i.e., relationships and executed database queries during runtime. The visualization follows the representation of communication within the landscape-level perspective. Thus, they are shown as edges (colored ) between tables and the thickness correlates with the current amount of executed queries. Once a table is opened, the edge connects the relevant columns (e.g., foreign keys in a table and their corresponding counterpart as a primary key in another table) instead of the tables (shown in Figure 7.19b). Hence, the operator can directly perceive the relation between two tables. In order to maintain a good readability, we plan to employ edge crossing reduction layout algorithms. If crossing edges could not be avoided, applying bundled edges offers an alternative.

In order to get details for database queries (between tables), the operator can click on a communication edge. As a result, all queries, which contain the connected tables, are shown in a sortable list with detailed information like the number of executions, the response time, and the concrete executed statement. This feature is particularly interesting, when conducting database optimizations, finding performance issues [CSJ+16], or evaluating database performance [RSB11]. Another feature includes highlighting queries and tables. When the operator selects a specific query or edge, the involved tables are highlighted. Additionally, selecting a table or related edges (database queries) highlights involved objects for further analysis or comprehension. Finally, the operator is able to move back to the landscape-level perspective at any point within the visualization. In this way, he can review the software landscape or take a look at another database and vice versa.

## 7. Tool-based Analysis



(a) Database tables, relationships, and executed queries



(b) Details of the tables *Product* and *Category*

**Figure 7.19.** Mockup of the database-level visualization: tables, relationships, and executed queries in a single database (*Core (DB2)*).



## 7.4. Database Visualization

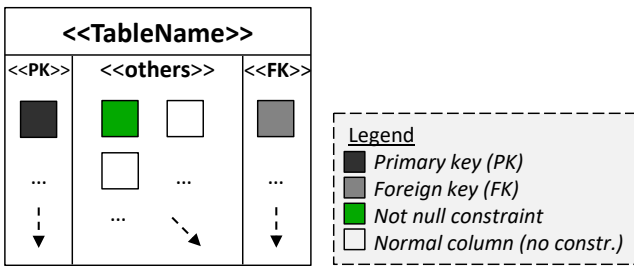


Figure 7.20. Semantics of an opened table: columns and constraints.

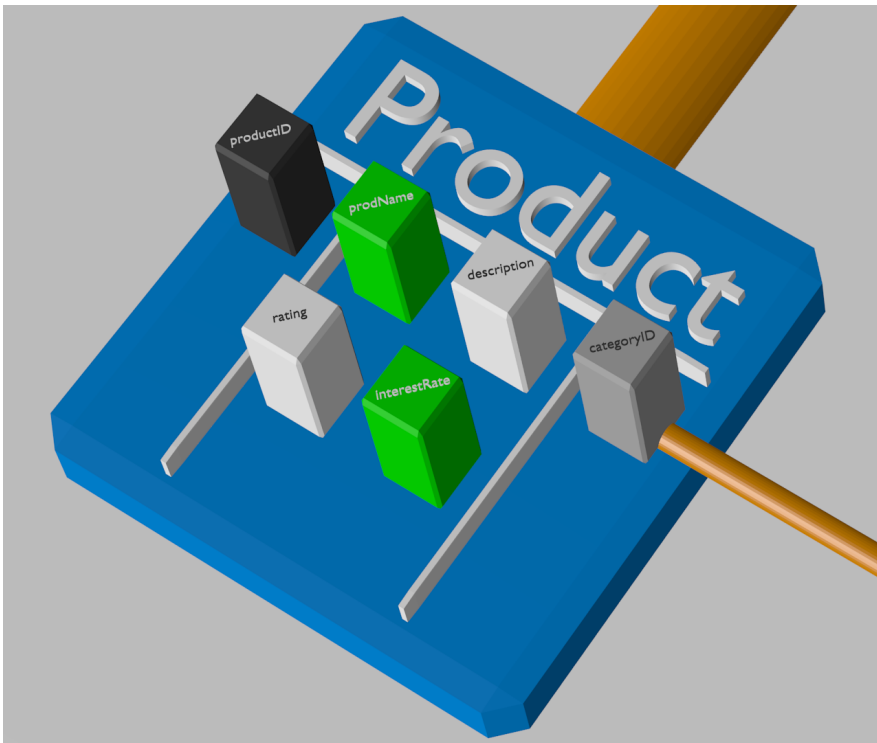


Figure 7.21. Visualized Opened table *Product* – showing columns, constraints, and relationships.



**Part III**

# **Evaluation**



# Evaluation Overview

The goals G1 to G3 will be evaluated with several case studies divided into three evaluation categories, one table for each category. Some of these case studies evaluate multiple goals. Thus, we present three tables of conducted case studies and related goals in the following. In order to understand the tables, we again list the previously mentioned goals:

- ▷ G1: Provide an approach to support modernization and modularization processes in open source research projects
- ▷ G2: Find and employ alternative display and interaction concepts for system and program comprehension using the software city metaphor
- ▷ G3: Research and evaluate alternative visual approaches for system and program comprehension

In Table 8.1, we present our conducted case studies and their related goals applying our *CORAL* approach. Afterwards, we illustrate our conducted case studies regarding the developed extensions as part of confirming the extensibility of our *ExplorViz* project in Table 8.2. Finally, in Table 8.3 we present our conducted case studies for our database comprehension approach. Furthermore, we evaluate the application of *ExplorViz* as a tool for modularization, modernization, and system and program comprehension tasks with other projects.

## 8. Evaluation Overview

Our first evaluation category evaluates our *CORAL* approach by several conducted case studies as shown in Table 8.1. We successfully applied multiple iterations of the *CORAL* approach on the open source research project *ExplorViz*. The related case studies (CS1.1 - CS1.4) are described as a comprehensive case study in Chapter 10. Furthermore, we performed two case studies (CS1.5 and CS1.6) to evaluate the software quality after the first, respectively second iteration of our approach's application within master's course lecture exercises with students. The students used several software quality metrics provided by the established related tools *SonarCloud*<sup>1</sup> and *Sonatype*<sup>2</sup>. Both tools were used to compare an initial version before an iteration with an improved version after an iteration. Both case studies showed an improvement of the software system *ExplorViz* and thus confirmed the performed successful modularization and reengineering processes.

---

<sup>1</sup><https://sonarcloud.io>

<sup>2</sup><https://www.sonatype.com>

**Table 8.1.** Evaluation: conducted case studies and their related goals applying CORAL.

Case Study Id	Name	Goals			Reference
		G1	G2	G3	
<i>CORAL Application</i>					
CS1.1	ExplorViz Modularization Motivation	✓			[ZKH18] <sup>PP</sup>
CS1.2	Two Iterations applied to ExplorViz	✓			[ZKH19c] <sup>PP</sup>
CS1.3	Two Iterations applied to ExplorViz	✓			[ZKH19b] <sup>TR</sup>
CS1.4	Three Iterations applied to ExplorViz	✓			[ZKH20] <sup>PP</sup>
CS1.5	Software Quality Metrics Evaluation	✓			SA 17/18, <sup>LE</sup>
CS1.6	Software Quality Metrics Evaluation	✓			SEPDS 19/20 <sup>LE</sup>

PP peer-reviewed publication    TR technical report    LE lecture exercise    BA bachelor's thesis

MA master's thesis    DA diploma thesis

## 8. Evaluation Overview

Our second evaluation category, as shown in Table 8.2, evaluates the extensibility aspect of our *CORAL* approach by several conducted case studies. While we successfully applied multiple iterations of the *CORAL* approach on the open source research project *ExplorViz*, we also developed a series of supplementary extensions for the project. Some of the related case studies (CS2.1 - CS2.9) are described as developed extensions within the comprehensive case study in Chapter 10. Every listed case study within the table represents a successfully developed extension, which provides new, useful features to *ExplorViz*. More precisely, every listed case study does not only address our primary research goal G1, but also research goal G2 or G3 in some cases, based on their added functionality to *ExplorViz*. An example are four consecutive case studies (CS2.4.1 - CS2.4.4), which evaluate the presented *ExplorViz VR* approach. Furthermore, each listed case study has been evaluated through at least a quantitative empirical evaluation. Finally, some of the developed extensions have been integrated into the core project based on their high impact for several users.



**Table 8.2.** Evaluation: conducted case studies and their related goals for developed extensions.

Case Study Id	Name	Goals			Reference
		G1	G2	G3	
<i>Developed Extensions</i>					
CS2.1	Architecture conformance checking	✓		✓	[Hac18] <sup>DA</sup>
CS2.2	Architecture runtime comparison	✓		✓	[Weg18] <sup>MA</sup>
CS2.3	Software architecture comparison	✓		✓	[Teu19] <sup>MA</sup>
CS2.4.1	VR Approach	✓	✓	✓	[Häs17] <sup>BA</sup>
CS2.4.2	VR Approach Gestures	✓	✓	✓	[ZKH19a] <sup>TR</sup>
CS2.4.3	VR Multi-User Usability	✓	✓	✓	[Han18; Kön18] <sup>BA</sup>
CS2.4.4	VR Multi-User Applicability	✓	✓	✓	[Brü20] <sup>BA</sup>
CS2.5	Eye-Tracking Program Comprehension	✓	✓	✓	[Kan17] <sup>MA</sup>
CS2.6	Brain Computer Interface	✓	✓	✓	[Möl17] <sup>BA</sup>
CS2.7.1	Application Discovery extension	✓			[Kra18] <sup>MA</sup>
CS2.7.2	Application Discovery integration	✓			[KZH18] <sup>PP</sup>
CS2.8	Tutorial	✓			[Mül19] <sup>MA</sup>
CS2.9	Dashboard	✓			[Kri19] <sup>BA</sup>

PP peer-reviewed publication    TR technical report    LE lecture exercise    BA bachelor's thesis

MA master's thesis    DA diploma thesis

## 8. Evaluation Overview

Finally, our third and last evaluation category, as shown in Table 8.3, evaluates the previously presented database visualization approach *RACCOON* (see Section 7.4) by two conducted case studies (CS3.1 and CS3.2). Furthermore, we successfully applied *ExplorViz* in several research and industrial collaboration projects. The related case studies (CS4.1 - CS4.6.2) describe the application of *ExplorViz* for modularization or modernization, or system and program comprehension tasks. Within Case Study CS4.4, students within our bachelor's course *software project* employed *ExplorViz* to understand a given software system, which should be extended by an anomaly detection service. The majority of students were able to apply *ExplorViz* and thus extend the software in the expected manner.

**Table 8.3.** Evaluation: conducted case studies and their related goals for database comprehension and employing *Explor Viz.*

Case Study Id	Name	Goals			Reference
		G1	G2	G3	
<i>Database Comprehension</i>					
CS3.1	Database monitoring extension	✓			[Zir17] <sup>PP</sup>
CS3.2	Database (behavior) visualization	✓	✓	✓	[ZH19] <sup>TR</sup>
<i>Employing Explor Viz as a tool</i>					
CS4.1	EPrints (Bottleneck visualization)	✓			[ZHC15] <sup>PP</sup>
CS4.2	EPrints (Bottleneck visualization)	✓			[ZHF+15] <sup>TR</sup>
CS4.3	Symphony Project	✓			[Eic17] <sup>MA</sup>
CS4.4	Anomaly Detection Service	✓			Software Project 16 <sup>LE</sup>
CS4.5.1	iObserve application	✓			[HJZ+17] <sup>PP</sup>
CS4.5.2	iObserve application	✓			[HZJ17] <sup>PP</sup>
CS4.6.1	Microservice Decomposition	✓			[Len19] <sup>MA</sup>
CS4.6.2	Microservice Decomposition	✓			[KZH+20] <sup>PP</sup>

<sup>PP</sup> peer-reviewed publication    <sup>TR</sup> technical report    <sup>LE</sup> lecture exercise

<sup>BA</sup> bachelor's thesis    <sup>MA</sup> master's thesis    <sup>DA</sup> diploma thesis



# *ExplorViz* VR Approach Evaluation

In this chapter, we present an evaluation of our VR approach *ExplorViz VR*. We split the evaluation into two parts, namely the evaluation of the single-user VR approach and the improved multi-user VR approach. First we define the goals for our evaluations in Section 9.1. Afterwards, we empirically evaluate our implementation with a focus on usability in Section 9.2. In Section 9.3, we present two empirical evaluations of the multi-user VR approach – the first focusing on the (perceived) usability of the approach for users, and the second focusing on its applicability with a qualitative and quantitative evaluation for system and program comprehension tasks, especially in teams. Finally, in Section 9.4, we summarize the results of the presented evaluations.

## 9. *ExplorViz* VR Approach Evaluation

### Previous Publications

Parts of this chapter are already published in the following works:

1. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
3. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
4. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49
5. Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, and Dan Kröger. “Microservice Decomposition via Static and Dynamic Analysis of the Monolith.” In: *Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2020

## 9.1 Goals

For the evaluation of our Virtual Reality approach *ExplorViz VR*, we focus on three aspects of our implementation, which we evaluate through several conducted lab experiments. Again, we follow the GQM approach [BW84; SB99; SBC+02] to define our evaluation goals, which are explained in the following.

### **VR-G1: Perceived Usability of the Single-User VR Approach**

Our first implementation of the single-user VR approach provides several new interaction and visual capabilities to explore software systems in VR. Thus, we need to investigate, if the approach is accepted by (first-time) users and if it could offer an alternative to system and program comprehension using *ExplorViz* in the usual way using traditional input devices and a screen. Additionally, we investigate the perceived user experience and usability, the latter especially in the aspect of navigation and interaction.

### **VR-G2: Perceived Usability of the Multi-User VR Approach**

By improving our single-user VR approach towards a collaboratively usable, multi-user VR approach, we also want to investigate the perceived usability of users. Thus, we are interested in the usability of the VR multi-user mode in general, the intuitiveness and usability of the controls and the usefulness of visual illustrations and of the spectator feature.

### **VR-G3: Applicability for System and Program Comprehension using the Multi-User VR Approach**

Another evaluation goal addresses the applicability for system and program comprehension tasks using the multi-user VR approach. More precisely, we want to investigate, if the approach is suitable for solving complex tasks in the context of static and dynamic software analysis.

## 9. *ExplorViz* VR Approach Evaluation

### **VR-G4: Applicability for Collaborative System and Program Comprehension using the Multi-User VR Approach**

Finally, our last evaluation goal is an extension of our previous goal. While solving complex comprehension tasks in general is an important use case of the approach, we are especially interested, if the approach is also applicable when solving complex comprehension tasks collaboratively in teams.

## **9.2 Case Study: Single-User VR Usability**

After we presented an overview of the implementation of the single-user VR approach, we conducted an empirical evaluation focusing on the usability of the approach in [Häs17]. We describe our experimental design and present the operation, results, discussion, and threats to validity. Finally, we conclude our evaluation by summarizing our results.

### **Experimental Design**

We based our evaluation on general software engineering experimentation guidelines [KPP+02; KDJ04; JG12; WRH+12] and conducted an usability study where subjects solved basic tasks while applying our VR approach. Once the subjects finished the tasks, they were asked to fill out a questionnaire in order to collect their experience. Furthermore, we (the usability study leaders) also made notes during the usability study to document our observations.

### **Research Questions**

We formulate the following three research questions (RQ-SU) for the usability study:

- ▷ RQ-SU1: Does employing the VR approach not restrain the user (User Experience)?
- ▷ RQ-SU2 Is the navigation within the VR approach well accepted by the user (Navigation)?



## 9.2. Case Study: Single-User VR Usability

- ▷ RQ-SU3: Is the controller-based interaction well accepted by the user (Interaction)?

In order to answer our research questions, we prepared tasks focusing on the navigation, interaction, and usability of our single-user VR approach.

### Hardware Configuration

For our usability study, we employed a single physical machine shown in Table 9.1. The employed HMD (HTC Vive) uses two Vive controllers and two base stations. The related play area is set to an area of approximately 14.4 m<sup>2</sup>.

**Table 9.1.** Hardware setup employed for the multi-user VR approach evaluation.

Machine	
CPU	Intel Core i5-6500
GPU	NVIDIA GeForce GTX 1070
RAM	16 GB
HMD	HTC Vive

### Software Configuration

The employed machine uses Microsoft Windows in Version 10 Pro - 64 Bit as operating system. Additionally, it runs an instance of the *ExplorViz* backend<sup>1</sup>, *ExplorViz* frontend<sup>2</sup> and has the *ExplorViz* frontend VR extension<sup>3</sup> installed.

For the usability study, we employed the *demo landscape* provided by *ExplorViz* in Version 1.2.0.<sup>4</sup> Basically, this landscape contains a small, mocked software landscape representing the open source research project PubFlow [BH13] (landscape-level) as shown in Figure 9.1. Furthermore, this

<sup>1</sup><https://github.com/ExplorViz/explorviz-backend/releases/tag/1.2.0>

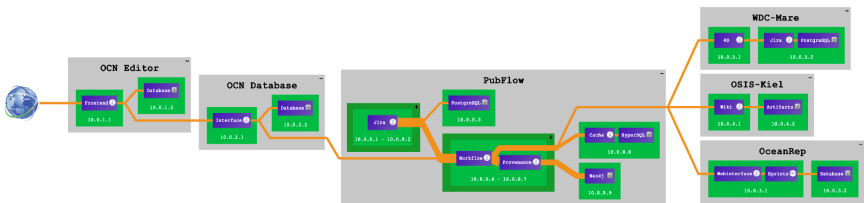
<sup>2</sup><https://github.com/ExplorViz/explorviz-frontend/releases/tag/1.2.0>

<sup>3</sup><https://github.com/ExplorViz/explorviz-frontend-extension-vr/releases/tag/v1.1>

<sup>4</sup><https://github.com/ExplorViz/explorviz-frontend/releases/tag/1.2.0>

## 9. ExplorViz VR Approach Evaluation

landscape also contains an explorable Application *Neo4J* (application level), a simplified version of the monitored graph database management system Neo4j, which can be seen in Figure 9.2. In order to answer our research questions, we prepared basic tasks focusing on the navigation and interaction of our VR approach. More precisely, we assigned geometrical-oriented tasks and were not interested in content-oriented tasks, i.e., problem-solving or comprehension.



**Figure 9.1.** Landscape-level perspective showing the *PubFlow* demo landscape of *ExplorViz* in Version 1.2.0.

### Population

For the usability study we were able to motivate 14 subjects (students and researchers) with a variety in prior VR experience and software visualization to participate.

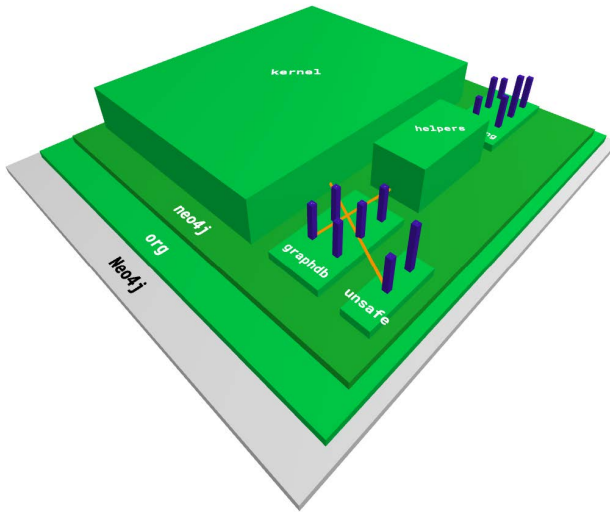
### Operation

In the following, we describe the operation of our usability study.

### Questionnaire

We employed a two-parted, paper-based questionnaire, because we wanted to provide a comfortable and well-known environment for the subjects. The

## 9.2. Case Study: Single-User VR Usability



**Figure 9.2.** Application-level perspective showing the simplified Neo4J application within the demo landscape of *ExplorViz* in Version 1.2.0.

first part addressed general information about the subject and was filled out at the beginning of the usability study. Afterwards, we conducted the actual usability study and the subjects completed the second part, which contained several questions focusing on the usability of the VR approach. We were especially interested in the perceived user experience while using the VR approach, and the intuitiveness of the navigation and finally of the interaction. For most of the questions the subjects were asked to answer them based on a 5-Point Likert Scale [Lik32] ranging from 4 (Strongly agree) to 0 (Strongly disagree). The questionnaire can be found in Appendix A.1. For further information about the usability study we refer to [Häs17].

### Pilot Study

Before we started the actual usability study, we conducted a small scale pilot study with colleagues as subjects first. Based on the received feedback we improved the assigned tasks and questionnaire.

## 9. *ExplorViz* VR Approach Evaluation

### **Training Phase**

In order to bring all subjects to a similar knowledge level, we conducted a training phase to get them familiar with the HMD, the navigation within the virtual space, and the controller-based interaction. The training took place within a limited perimeter within the virtual space and covered the navigation and interaction capabilities step by step.

### **Usability Study Phase**

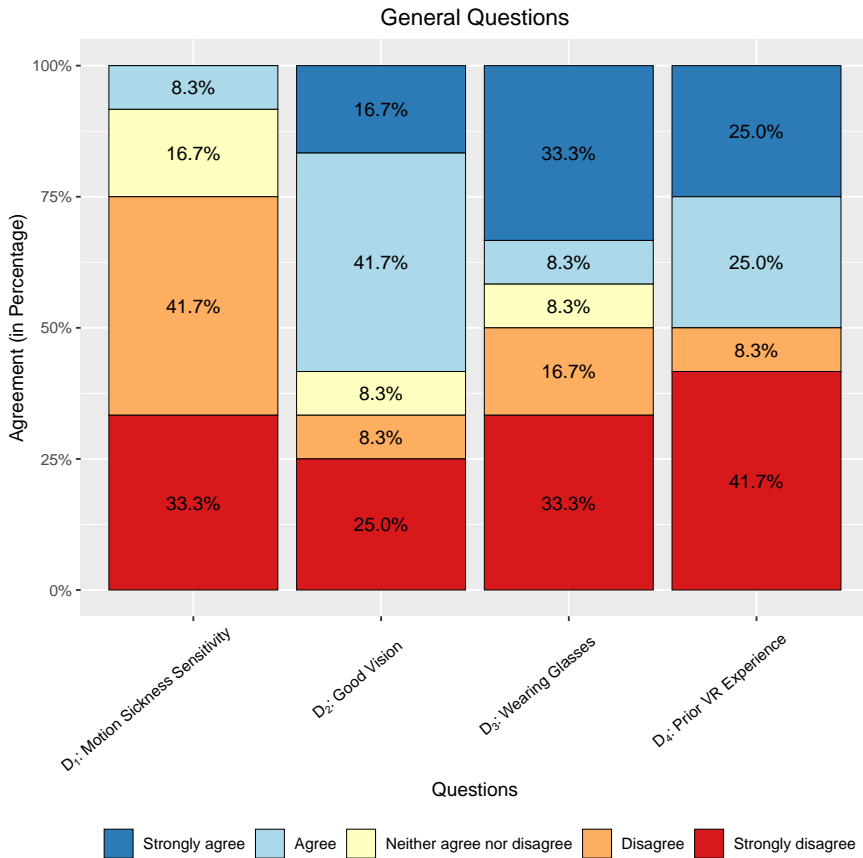
After finishing the training phase, the actual usability study began. Since we were interested in the usability, the subjects performed the previously learned navigation and interaction capabilities provided by us orally in a specific order.

## **Results**

### **General Questions**

Since there exist several restrictions, which affect the usability for a user within a VR environment, we asked some related questions upfront. The results are shown in Figure 9.3. Motion or sea sickness plays an important role when using VR. Fortunately, 75% of our subjects were not affected by this circumstance ( $D_1$ ). Only one subject replied that he has a strong vulnerability. Having a reducing vision ability ( $D_2$ ), also influences the usability, especially with regard to spot objects and read text labels within our visualization. About 40% of our subjects stated, that they could see very well or well. Consequently, we asked if a subject wears glasses ( $D_3$ ), which affects the vision as well, regardless of whether they are short-sighted or long-sighted. About a third wear glasses the whole time, another third often to sometimes, and the last third does not wear glasses. Our last question addressed the prior experience of a subject with VR ( $D_4$ ). More than half of the subjects has some or even much experience with VR, while about the remaining half of our subject has no VR experience at all.

## 9.2. Case Study: Single-User VR Usability

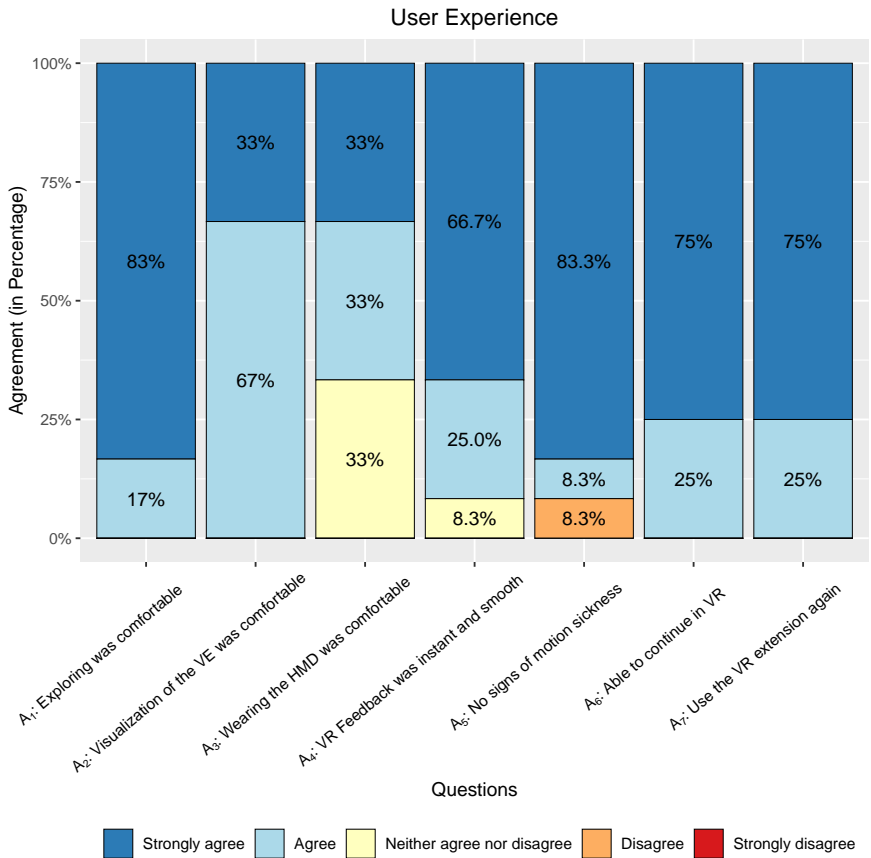


**Figure 9.3.** Usability study results for *General Questions* ( $D_1$ - $D_4$ ).

### User Experience

The *User Experience* is depending on the quality of the visualization and wearing comfort of the employed HMD. Both aspects affect the sense of well-being for a user. We asked the subjects a set of statements, which addresses these aspects. The results are shown in Figure 9.4. The average agreement

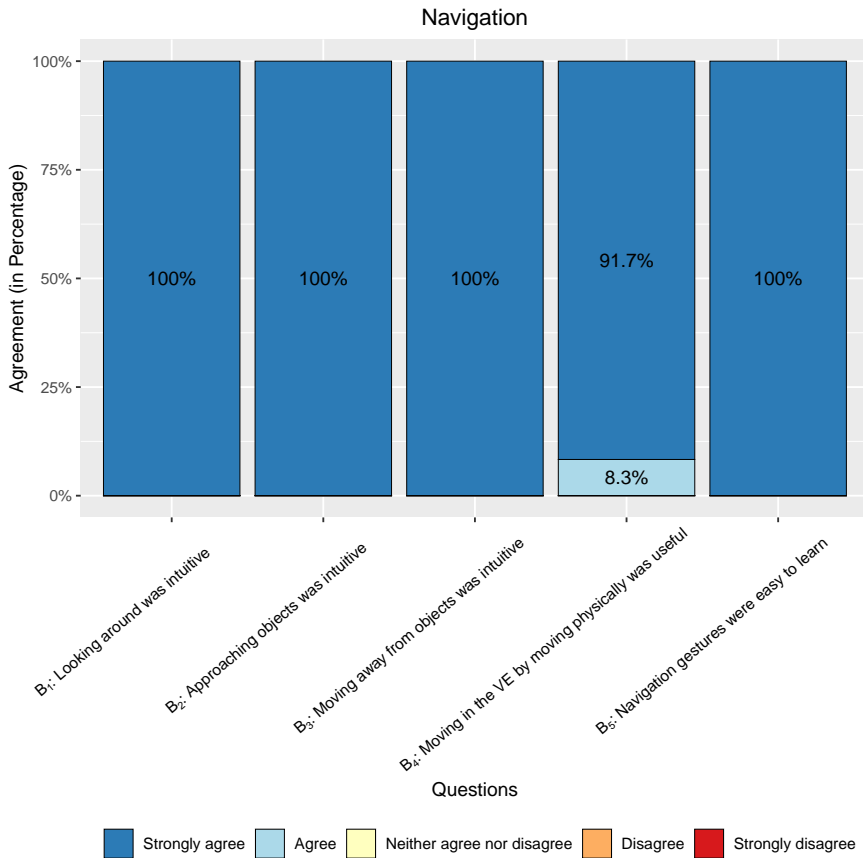
## 9. ExplorViz VR Approach Evaluation



**Figure 9.4.** Usability study results for *User Experience* questions ( $A_1$ - $A_7$ ).

rate is 89%. More precisely, we received a high, positive agreement rate for a majority of the questions ( $A_1$ ,  $A_2$ ,  $A_6$ ,  $A_7$ ). Only the questions  $A_3$ ,  $A_4$ , and  $A_5$  received a slightly lower rating. One subject noted that he felt a sign of motion sickness.

## 9.2. Case Study: Single-User VR Usability



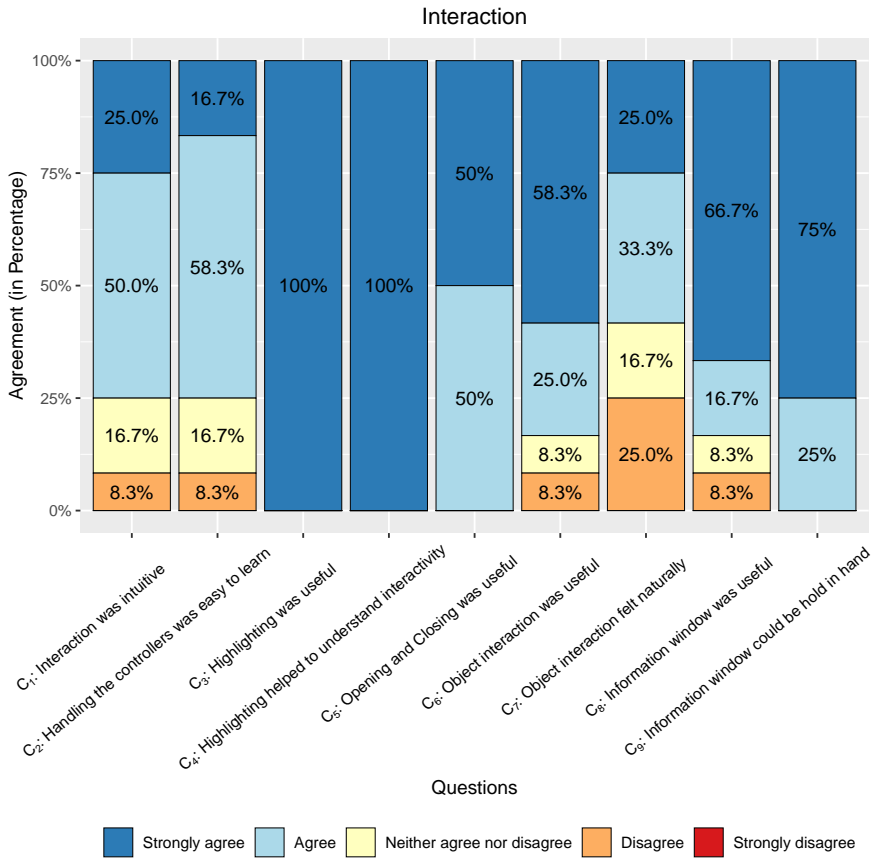
**Figure 9.5.** Usability study results for *Navigation* questions ( $B_1$ - $B_5$ ).

### Navigation

The *Navigation* is implemented by the HMD and related head and body movements. Looking around in the virtual environment and moving in it, especially by moving physically has a significant influence on the *Navigation* experience for a user. In Figure 9.5, we present the results for this aspect.

## 9. ExplorViz VR Approach Evaluation

With the exception of Question  $B_4$ , all questions of this category were rated with an optimal agreement rate of 100%.



**Figure 9.6.** Usability study results for *Interaction* questions ( $C_1$ - $C_9$ ).



## 9.2. Case Study: Single-User VR Usability

### Interaction

The *Interaction* of our VR approach is realized by two controllers, bundled by the HMD, and movements of the body within the virtual environment. In order to offer a suitable interaction for a user, it is important that related gestures are easy to learn. Especially the usability and reasonable application of the gestures influence the *Interaction* experience for a user. The results are shown in Figure 9.5. The average agreement rate is 84.25%. Particular questions  $C_3$  and  $C_4$  received an optimal agreement rate of 100%. Furthermore, the Questions  $C_1$ ,  $C_2$ ,  $C_6$ , and  $C_8$  got disagreed by only one subject, and Question  $C_7$  by three subjects.

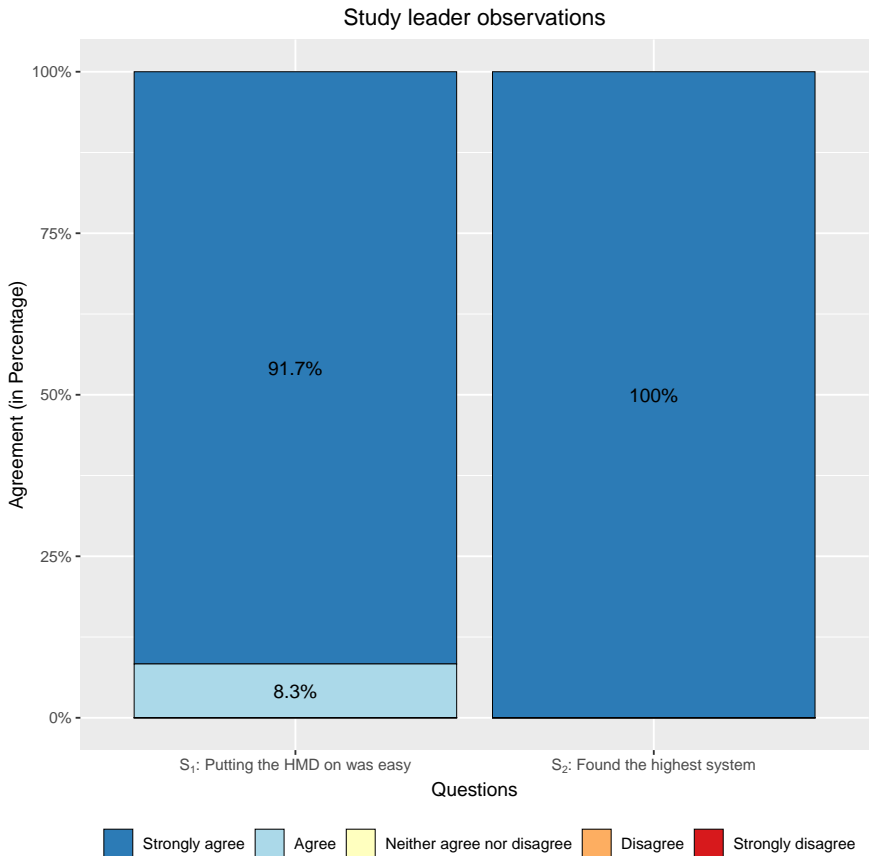
### Discussion

In this section we analyze and discuss the results of our previously presented VR usability study. Thus, we start by presenting our observations as a study leader depicted in Figure 9.7. First, putting on the HMD ( $S_1$ ) was an easy task for all subjects, only in one case we needed to perform some adjustments for a good fit. Second, the subjects had to identify the highest visualized system, with regard to its geometrical extension ( $S_2$ ). All subjects were able to find and name the correct system offhand. The results allow us to draw the conclusion that applying the third dimension can be handled by the users very well. In the following, we present the analysis and discussion of our three question categories – *User Experience*, *Navigation*, and *Interaction*.

### User Experience

While exploring the virtual environment was (very) comfortable for all subjects ( $A_1$ ), the visualization of the virtual environment was not rated that high ( $A_2$ ). Two thirds of our subjects stated it comfortable, but not very comfortable. A possible explanation might be limited resolution of the employed HMD or the slightly blurry text labels. The comfort of wearing the HMD ( $A_3$ ) only received an average rating. One reason for this could be that the HMD might have caused small pressure points for some subjects. Another reason might be the cable that hindered the subjects in their

## 9. ExplorViz VR Approach Evaluation



**Figure 9.7.** Usability study leader observation results ( $S_1$ - $S_2$ ).

movement. We noticed, that only subjects without prior VR experience stated, that they were disturbed by the cable. Our own experience confirms that while using the HMD the cable is a hindrance for the first few times and one has to get used to it after a short while. Our question focusing on an instant and smooth feedback from the virtual environment ( $A_4$ )

## 9.2. Case Study: Single-User VR Usability

was evaluated very well by two thirds, well by one quarter, and neutral by one subject. We suspect that the tracking of the HMD and controllers got a time lag for some of the subjects or loading times when opening or closing packages within the visualization. Only one subject recognized signs of motion sickness ( $A_5$ ). The same subject previously stated that he has a related vulnerability. Thus, we imply that the usage of our VR approach does not cause motion sickness to users, unless they already have a vulnerability. Furthermore, the results for the ability to continue using the VR approach after the usability study ( $A_6$ ) and if the subject would use the VR approach again ( $A_7$ ) were evaluated well. In both cases three out of four subjects agreed fully, the remaining quarter just agreed. We therefore assume, that our VR approach offers a comfortable virtual environment, which is worth to being applied again. Overall, we received an average agreement rate of 89%, which states an excellent evaluation of the perceived *User Experience* of our VR approach. While analyzing the usability study results, we did not recognize that differences regarding the sense of sight, wearing glasses, sensitivity for motion sickness, or prior VR experience within the subject group have a greater effect on the *User Experience*. However, missing a prior VR experience caused one of four subjects to get disturbed by the physical cable of the HMD. Thus, we can state that a missing VR experience and sensitivity for motion sickness have a slight influence on the *User Experience*. Finally, we did not identify obstacles based on the results, which could influence the categories *Navigation* or *Interaction* upfront. This observation allows us to draw the conclusion applying our VR approach does not restrain the user and thus provides an answer to Research Question *RQ-SU1*.

### Navigation

Our questions within the category *Navigation* address the Intuitiveness ( $B_1$ - $B_3$ ), Practicability ( $B_4$ ), and Learnability ( $B_5$ ) of our VR approach. Since all questions of this category were rated with an optimal agreement rate of 100%, except for Question  $B_4$ , we imply that the implementation of the *Navigation* within our VR approach is very intuitive. While analyzing the usability study results, we did not observe that differences regarding the

## 9. *ExplorViz* VR Approach Evaluation

sense of sight, wearing glasses, sensitivity for motion sickness, or prior VR experience within the subject group have an effect on the *Navigation*. The results allow us to draw the conclusion that the *Navigation* within our VR approach is very well accepted by users and thus gives us a positive answer towards Research Question *RQ-SU2*.

### **Interaction**

In order to analyze the results of the *Interaction* category, we split the related questions into the three groups Intuitiveness ( $C_1, C_7, C_9$ ), Practicability ( $C_3, C_5, C_6, C_8$ ), and Learnability ( $C_2, C_4$ ). The question, if the interaction with the virtual environment is intuitive ( $C_1$ ) was rated only with an average agreement. The same applies for rating how easy it is to learn the handling of the controllers ( $C_2$ ). Based on our observations we can state, that deleting an application within the virtual environment caused a problem for one third of the subjects and led to a slight confusion. We encountered this state by giving some instructions to get them back on the track again, but this might explain the low rating. The automatic highlighting of interactive objects by using the controller ray ( $C_3$ ) was rated with an optimal agreement rate of 100%. The same applies for Question  $C_4$ , where we asked if the highlighting helps to understand which objects are interactive. Opening and closing packages ( $C_5$ ) was stated as useful or even very useful. Interacting with objects by rotating, moving, or zooming in and out ( $C_6$ ) got an average rating of 83.25%, which is still a good value. Using the trigger button for this gesture might have been a better choice since the gesture itself constitutes a grasping movement. We also asked, if the interaction with objects felt natural ( $C_7$ ), based on the grasping gesture, i.e., grabbing the object with the hand. We received only an average agreement rate of 64.5%, which leaves room for improvement. One reason might be, that the virtual objects could be too far away in some cases to be grabbed or held within the hand. A solution could be to adapt the texture of the visualized controller towards an interactive object. The last two questions addressed the information window which is bound to the controller of a user within the visualization. The window shows additional information about a selected object next to the visualized controller. The visualization of the window itself and its use-

## 9.2. Case Study: Single-User VR Usability

fulness ( $C_8$ ) were rated with an average agreement of 85.5%. The handling of the window and the ability to adjust the visual angle like interacting with a newspaper ( $C_9$ ) were rated positively by all subjects. Overall, the *Interaction* category received an average agreement rate of 84.25%, which depicts a good result. The results allow us to draw the conclusion that the *Interaction* within our VR approach is very well accepted by users and thus gives us a positive answer towards Research Question *RQ-SU3*.

### Qualitative Feedback

In addition to our presented results we asked for qualitative feedback to improve our VR approach. In the following, we present the most frequently stated answers. About a quarter of the subjects would welcome a sharper text within the visualization, since reading the text labels despite moving closer was not sufficient in some cases. Tweaking the rendering of the text labels could address this concern. Two subjects noted, that the overall sharpness of the visualization could be enhanced. In this regard we are bound to the technical limits of the employed HMD. Although we used the Vive, which features a higher resolution than previous devices, there is potential for improvement. Thus, a follow-up model HMD with a higher resolution could be a solution. Another aspect mentioned by three subjects was that the cable hindered their physical movement. A workaround may be to attach the cable to the ceiling, but this solution does not completely resolve the issue. Similarly, a successor HMD with a built-in tracking capability could offer a comprehensive solution.

### Threats to Validity

In this section, we discuss the threats to both internal and external validity that might have influenced our results [WRH+12; SCC+02; JG12].

#### Internal Validity

**Training Phase** In order to bring all subjects to a similar knowledge level, we conducted a training phase to get familiar with the HMD, the navigation

## 9. *ExplorViz* VR Approach Evaluation

within the virtual space, and the controller-based interaction. We noticed during the usability study, that some subjects had problems to remember the controller mapping for navigating and interacting in the visualization. A possible cause might be the missing option to display the mapping during the usability study inside the visualization. If we would take this aspect into account and further enhance the training phase, we expect to improve future results.

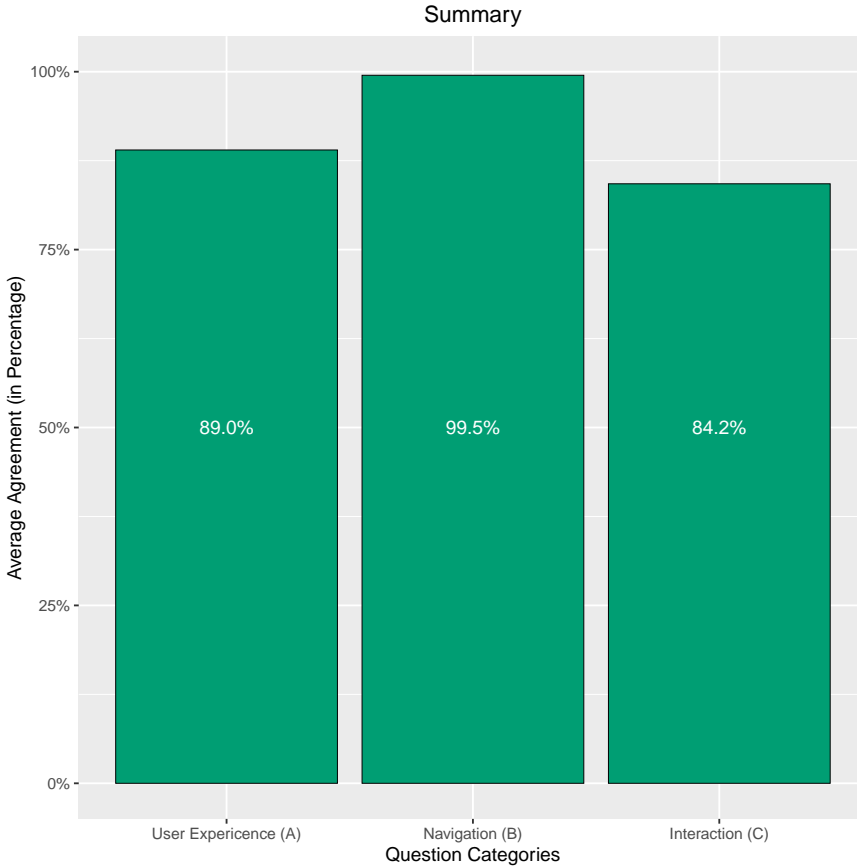
**Tasks** During the usability study we asked the subjects to perform simple navigation and interaction tasks. The chosen tasks might be not adequate in design or scope to comprehensively verify the user experience and usability of our VR approach.

### **External Validity**

**Subjects** As we had only a small number of subjects within the usability study, we cannot draw conclusions for a larger group of users. According to [NL93] a typical testing session identifies 31% of all usability problems in a design and 85% of a sites problems can be found with as few as five participants. Thus, the size of our test group is sufficient for identifying a large set of usability problems. Additionally, a small share of the subjects were at least familiar with one of the usability study leaders. Thus, this circumstance might have lead to an experimentator bias. However, in order to validate our results, we need to perform another usability study with a larger test group.

**Generalization** According to [SCC+02] experiment results cannot be generalized in every case. An example would be the dependency between our chosen tasks, the HMD, and the rendered software visualization. Thus, our results might not be applicable for other HMDs or visualizations. Another relevant aspect is the missing prior experience of VR for some of our subjects. Using VR for the first time is challenging and might influence our results as well. Hence, without further usability studies our results might not be applicable to generalization.

## Summary



**Figure 9.8.** Average agreement rates for the question categories A - C.

In this evaluation, we presented an usability study to verify the user experience and usability of our VR approach. We found answers to our research questions *RQ-SU1*, *RQ-SU2*, and *RQ-SU3* due to our conducted usability study. In Figure 9.8, the average agreement rate for our three

## 9. *ExplorViz* VR Approach Evaluation

question categories *User Experience*, *Navigation*, and *Interaction* are shown. Based on the average agreement rate of 89% for *User Experience*, we can confirm Research Question *RQ-SU1*, i.e., employing the VR approach does not restrain the user. The same applies to Research Question *RQ-SU2* (“Is the navigation within the VR approach well accepted by the user?”) with an almost optimal average agreement rate of 99.5%. Our last research question, *RQ-SU3* (“Is the controller-based interaction well accepted by the user?”), can be confirmed as well. However, we got a slightly lower average agreement rate of 84.2%.

In particular the *Navigation* of our VR approach was accepted very well and felt intuitive. The *User Experience* and *Interaction* aspects of our approach also received good evaluations, but leaves room for improvement. In combination with our observations shown in Figure 9.7, we imply that our VR approach combines the advantages of 3D visualizations with navigation and interaction capabilities in a virtual environment.

### 9.3 Case Study: Distributed Multi-User VR Environments

#### 9.3.1 Usability and User Experience Evaluation

After we presented a usability evaluation of our single-user VR approach, we conducted an empirical evaluation focusing on the usability and user experience of our multi-user VR approach in [Han18; Kön18]. We describe our experimental design and present the operation, results, discussion, and threats to validity. Finally, we conclude the evaluation by summarizing the results.

#### Experimental Design

We based our evaluation on general software engineering experimentation guidelines [KPP+02; KDJ04; JG12; WRH+12] and conducted an usability where subjects in teams of two solved basic tasks while applying our VR



### 9.3. Case Study: Distributed Multi-User VR Environments

approach. Once the subjects finished the tasks, they were asked to fill out a questionnaire in order to obtain their experience. Furthermore, we (the usability study leaders) also made notes during the usability study to document our observations.

#### Research Questions

We formulate the following four research questions (RQ-MU-A) for the usability study:

- ▷ RQ-MU-A1: Does the VR multi-user approach provide a good usability in general (User Experience)?
- ▷ RQ-MU-A2: Is the controller-based interaction well accepted by the user (Interaction)?
- ▷ RQ-MU-A3: Are the visual illustrations useful for the user (Visual Illustration)?
- ▷ RQ-MU-A4: Is the ability to spectate a multi-user VR session useful for the user (Spectator)?

In order to answer our research questions, we prepared tasks focusing on the collaborative navigation and interaction of our multi-user VR approach.

#### Hardware Configuration

For our usability study, we employed three physical machines shown in Table 9.2. The Oculus Rift is employed with two Oculus Touch controllers (one for each hand) and three Oculus Sensors. The play area is configured to an almost quadratic area of approximately 4.2 m<sup>2</sup>. The HTC Vive uses two Vive controllers and two base stations. The related play area is set to an area of approximately 14.4 m<sup>2</sup>. Both HMDs are set up in the same physical room. Thus, the subjects of each team are able to communicate with each other while wearing their respective HMDs and controllers.

## 9. *ExplorViz* VR Approach Evaluation

**Table 9.2.** Hardware setup employed for the usability multi-user VR approach evaluation.

	Machine 1	Machine 2	Machine 3
CPU	Intel Core i5-6500		
GPU	NVIDIA GeForce GTX 1070	NVIDIA GeForce GTX 950	
RAM	16 GB		
HMD	Oculus Rift	HTC Vive	None

### Software Configuration

All three machines use Microsoft Windows in Version 10 Pro - 64 Bit as operating system. Machine 1 and 2 both run an instance of the *ExplorViz* frontend<sup>5</sup> and have each the *ExplorViz* frontend VR extension<sup>6</sup> installed. Machine 3 runs an instance of the *ExplorViz* backend.<sup>7</sup> Furthermore, the *ExplorViz* backend extension VR<sup>8</sup> is integrated, to which the other machines connect. For the usability study, we employed the *demo landscape* from our previous usability study as described in Section 9.2, which is provided by *ExplorViz* in Version 1.3.0.<sup>9</sup>

### Population

For the usability study we were able to motivate 22 subjects (students and researchers) and thus 11 teams with a variety in prior VR experience and software visualization to participate.

### Operation

In the following, we describe the operation of our usability study. Since evaluating the multi-user mode requires more than a single user, the usabil-

<sup>5</sup><https://github.com/ExplorViz/explorviz-frontend/releases/tag/v1.3.0>

<sup>6</sup><https://github.com/ExplorViz/explorviz-frontend-extension-vr/releases/tag/v1.3.0>

<sup>7</sup><https://github.com/ExplorViz/explorviz-backend/releases/tag/v1.3.0>

<sup>8</sup><https://github.com/ExplorViz/explorviz-backend-extension-vr/releases/tag/v1.3.0>

<sup>9</sup><https://github.com/ExplorViz/ExplorViz/releases/tag/v1.3.0>

### 9.3. Case Study: Distributed Multi-User VR Environments

ity study is performed by teams of two within a single session. One subject uses the Oculus Rift and the other one the HTC Vive. The choice of the device rested within the team.

#### Questionnaire

We employed a two-parted paper-based questionnaire, because we wanted to provide a comfortable and well-known environment for the subjects. The first part addressed general information about the subject and was filled out at the beginning of the usability study. Afterwards, we conducted the actual usability study and the subjects completed the second part, which contained several questions focusing on the collaborative usability of the VR approach. We were especially interested in the perceived user experience while using the multi-user VR approach, and the intuitiveness of the interaction. For most of the questions the subjects were asked to answer them based on a 4-Point Likert Scale [Lik32] ranging from 3 (Strongly agree) to 0 (Strongly disagree) without a neutral value. The questionnaire can be found in Appendix A.2. For further information about the usability study we refer to [Han18; Kön18].

#### Pilot Study

Before we started the actual usability study, we conducted a small scale pilot study with colleagues as subjects first. Based on the received feedback we improved the assigned tasks and questionnaire.

#### Training Phase

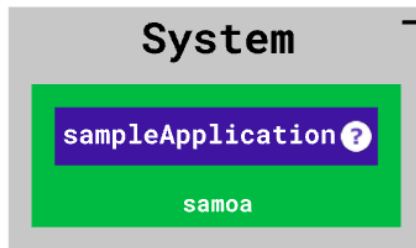
In order to bring all subjects to a similar knowledge level, we conducted a training phase to get them familiar with the HMD, the navigation within the virtual space, and the controller-based interaction. The training took place within a limited perimeter within the virtual space and covers the navigation and interaction capabilities step by step. For the training phase, we employed the *Sample Application*<sup>10</sup> provided by *ExplorViz* in combination

---

<sup>10</sup><https://github.com/ExplorViz/sampleApplication>

## 9. *ExplorViz* VR Approach Evaluation

with *ExplorViz* in Version 1.3.0.<sup>11</sup> Basically, this landscape contains a small, mocked software landscape representing a single, explorable application *SampleApplication* (landscape-level) as shown in Figure 9.9. The application *SampleApplication* (application level) is a minimalist Java application paired with a simple load driver, which randomly executes a reconfigured set of Java and JDBC methods Figure 9.10.



**Figure 9.9.** Landscape-level perspective showing the *SampleApplication* landscape visualized in *ExplorViz* in Version 1.3.0.

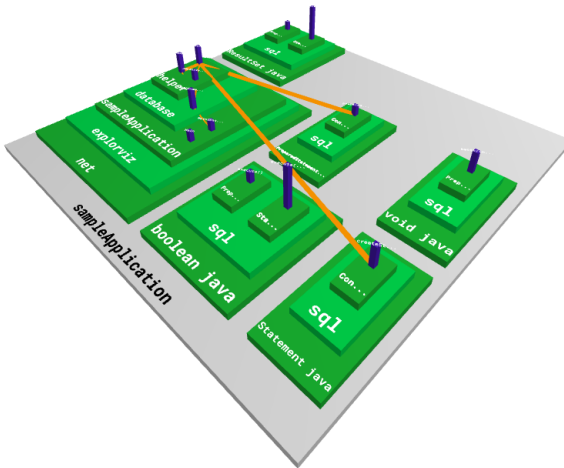
### Experiment Phase

After finishing the training phase, the actual usability study begins. Since we are interested in the usability, each team performed the previously learned navigation and interaction capabilities provided by us orally in a specific order to solve tasks collaboratively.

---

<sup>11</sup><https://github.com/ExplorViz/ExplorViz/releases/tag/v1.3.0>

### 9.3. Case Study: Distributed Multi-User VR Environments



**Figure 9.10.** Application-level perspective showing the minimalist application *SampleApplication* within the same named landscape visualized in *ExplorViz* in Version 1.3.0.

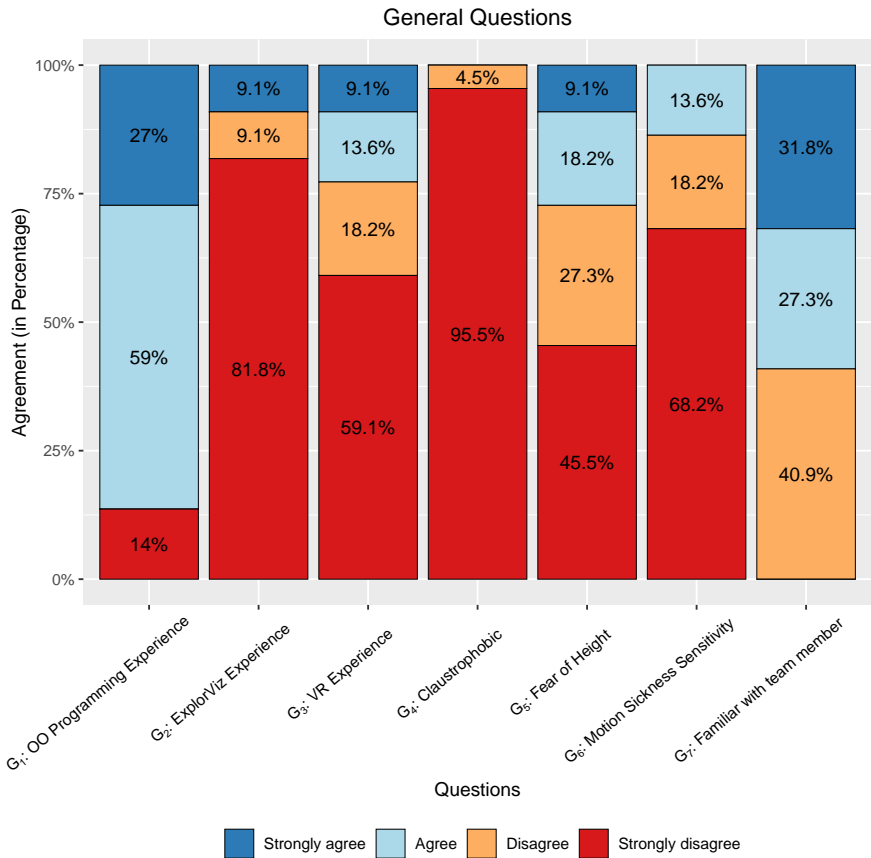
## Results

### General Questions

Since there exist several restrictions, which affect the usability for a user within a VR environment, we asked some related questions upfront. Thus, we were interested if our subjects wear glasses, which affects the vision as well. 41% of our subjects confirmed, that they wear glasses. Having a vision impairment, also influences the usability, especially with regard to spot objects, read text labels, or distinguish different colors within our visualization. About 46% of our subjects stated, that they are affected by a vision impairment such as being short or far sighted or having a red and green deficiency.

Furthermore, we were interested in prior related experiences, other restrictions, and how well the team members know each other in advance. The results are shown in Figure 9.3. More than 80% of our subjects stated that

## 9. ExplorViz VR Approach Evaluation



**Figure 9.11.** Usability study results for *General Questions* (G<sub>1</sub>-G<sub>7</sub>).

they have experience in object-oriented programming (G<sub>1</sub>). In comparison only about 20% of our subjects have prior experience with *ExplorViz* (G<sub>2</sub>). Our next question addressed the prior experience of a subject with VR (G<sub>3</sub>). About 23% of our subjects have some or even much experience with VR on the one hand, while the remaining subjects have no VR experience at

### 9.3. Case Study: Distributed Multi-User VR Environments

all. Since VR is experienced by people differently, we were also interested if additional restrictions exist. Consequently, we asked if a subject has a claustrophobic deficiency ( $G_4$ ) or a fear of heights ( $G_5$ ). Fortunately, no subject stated that he has a noticeable claustrophobic deficiency and only about 30% of the subjects are affected by fear of heights. Motion or sea sickness also plays an important role when using VR. Fortunately, about 86% of our subjects were not affected by this circumstance ( $G_6$ ). Our last question addressed the prior experience how well the team members know each other in advance ( $G_7$ ). More than half of the subjects stated that they know their team member at least well.

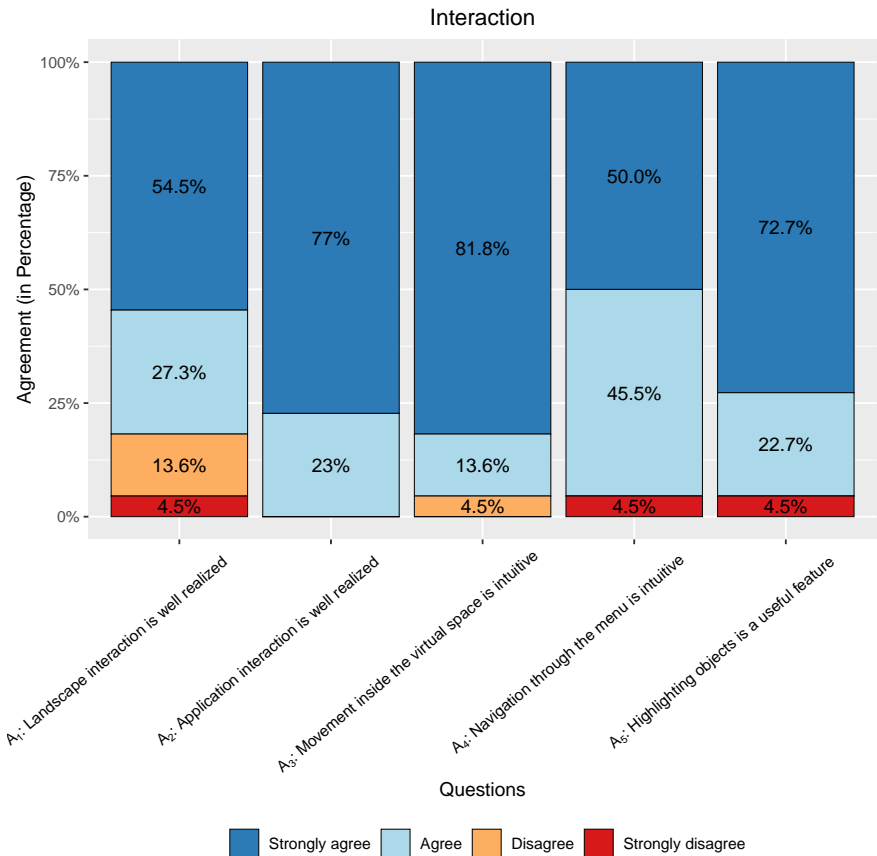
#### **Interaction**

The *Interaction* of our VR approach is realized by two controllers, bundled with each of our two HMDs, and movements of the body within the virtual environment. In order to offer a suitable interaction for a user, it is important that related gestures are easy to learn. Especially the usability and reasonable applicability of the gestures influence the *Interaction* experience for a user. The results are shown in Figure 9.12. The average agreement rate is about 94%. Particular questions  $A_3$ , and  $A_4$  received an agreement rate of about 95%. Especially Question  $A_2$  was rated only positively. Furthermore, the questions  $A_1$ ,  $A_3$ ,  $A_4$ , and  $A_5$  got strongly disagreed each by only a single subject.

#### **User Experience**

The *User Experience* is depending on the quality of the visualization and wearing comfort of the employed HMD. Both aspects affect the sense of well-being for a user. We asked the subjects a set of statements, which addresses these aspects. The results are shown in Figure 9.13. The average agreement rate is about 93%. More precisely, we received a high, positive agreement for the questions  $B_1$  and  $B_3$ . Especially Question  $A_1$  was rated only positively. Only the questions  $B_2$  and  $B_4$  received a slightly lower rating. One subject noted that he would definitely not use *ExplorViz* with the VR extension again, due to the experienced motion sickness.

## 9. ExplorViz VR Approach Evaluation



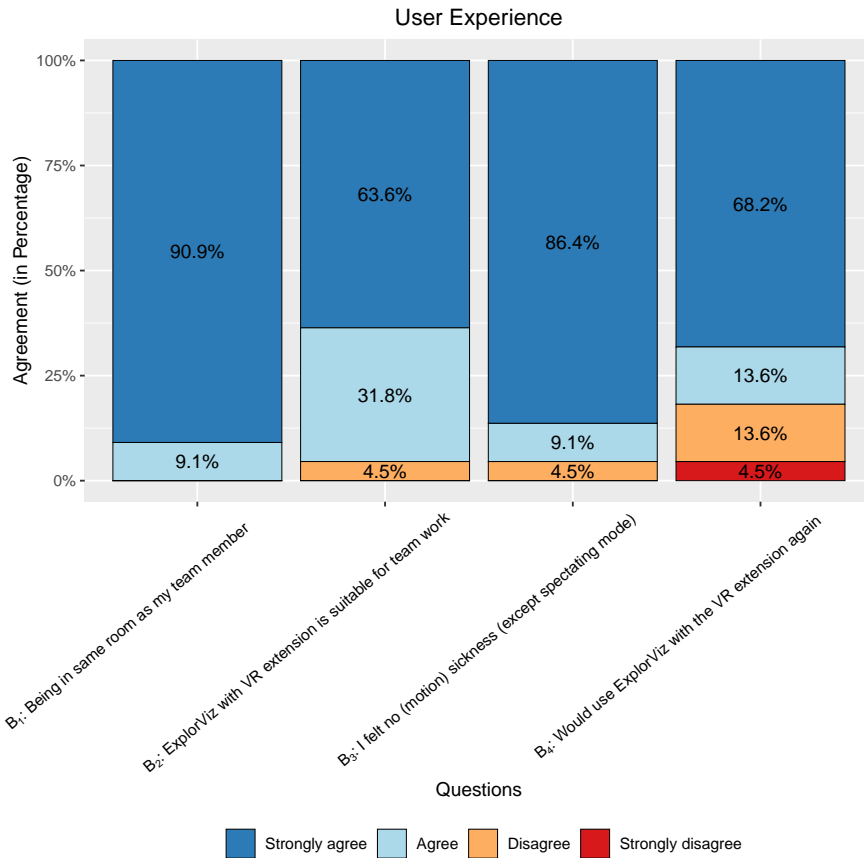
**Figure 9.12.** Usability study results for *Interaction* questions ( $A_1$ - $A_5$ ).

### Visual Appearance

We were also interested in the subjects' opinions on the implemented text overlays, including menus, and visualization of the other team member. In Figure 9.5, we present the results for this aspect. The average agreement rate is 91%. Especially Question  $C_4$  was rated only positively. Only one



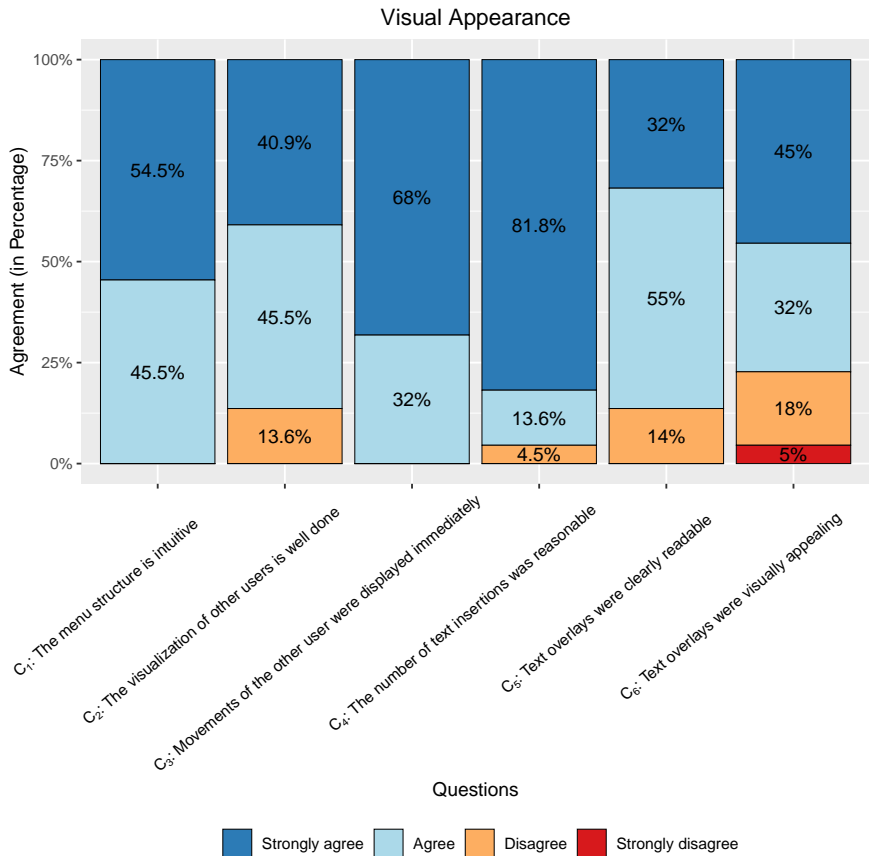
### 9.3. Case Study: Distributed Multi-User VR Environments



**Figure 9.13.** Usability study results for *User Experience* questions ( $B_1$ - $B_4$ ).

subject stated that he absolutely did not find the text overlays visually appealing. All other questions did not receive a single strong disagreement rating.

## 9. ExplorViz VR Approach Evaluation



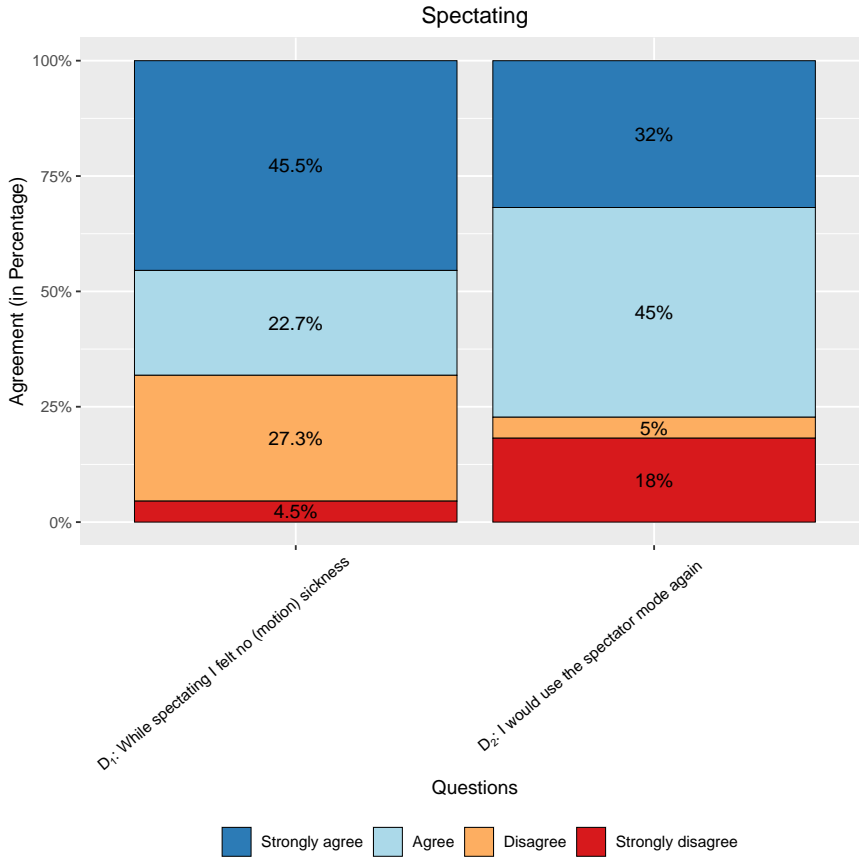
**Figure 9.14.** Usability study results for *Visual Appearance* questions (C<sub>1</sub>-C<sub>6</sub>).

### Spectating

Since spectating is a new, experimental feature within our VR approach, we were interested in feedback regarding this aspect in particular. While using the feature, the user loses control over their movements in VR and can experience the application from the visual perspective of another user.

### 9.3. Case Study: Distributed Multi-User VR Environments

Thus, we wanted to evaluate the perceived user experience and usefulness of this feature. The results are shown in Figure 9.15.



**Figure 9.15.** Usability study results for *Spectating* questions ( $D_1$  and  $D_2$ ).

For both questions ( $D_1$  and  $D_2$ ) we received a positive agreement rate of about 68%, respectively 77% of our subjects. However, 25% of our subjects would prefer not to use the spectating feature again.

## 9. *ExplorViz* VR Approach Evaluation

### Discussion

In this section we analyze and discuss the results of our previously presented multi-user VR usability study. In the following, we present the analysis and discussion of our four question categories – *Interaction*, *User Experience*, *Visual Appearance*, and *Spectating*.

### Interaction

As the interaction capabilities directly affect the usability of the multi-user approach, we were especially interested in the results. Moving and rotating the landscape ( $A_1$ ) received the lowest agreement (about 82%) within our *Interaction* category. One reason might be, that the landscape can only be moved and rotated with a sequence of single button presses. This states an uncomfortable interaction with a visualized software landscape. Another reason could be that moving the landscape is realized differently than moving an application. The interaction and movement of the application ( $A_2$ ) was rated as very intuitive. Therefore, it occurred that subjects tried to grab the landscape just like they could grab an application, which did not work. A solution could be to bring the movements into line. Furthermore, the movement of the landscape could be realized with the trackpad (HTC Vive) or with the virtual analog sticks (Oculus Rift) of the respective controllers. This would facilitate a smoother movement of a landscape instead of multiple button presses in a sequence. Regardless of the employed HMD, we utilized a token-based mechanism for performing translations on applications. Thus, only one subject could move it at a time to avoid conflicts. The movement through the virtual space ( $A_3$ ) was also rated as intuitive with an average rating of about 95%. The same applies for Question  $A_4$ , where we asked if navigating through the interactive menu was intuitive. However, we would consider to realize the navigation of the menu on the basis of the trackpad on the left controller as an alternative. This would allow a one-handed usage of the menu instead of using both controllers. The last question ( $A_5$ ) addressed the highlighting capability of objects within the visualization. Only one subject stated, that the feature is no useful in his opinion. Additionally, one subject mentioned that pointing with the ray of the controller on an object would be more intuitive. Another subject would

### 9.3. Case Study: Distributed Multi-User VR Environments

have preferred to highlight multiple objects, instead only a single one at a time. In our opinion highlighting multiple objects could lead to confusion, especially if many users are using this feature. Finally, we noticed that some subjects were puzzled that closed components cannot be highlighted. We plan to implement this feature in an upcoming version for enhanced usability. Overall, the *Interaction* category received an average agreement rate of about 96%, which depicts a good result. The results allow us to draw the conclusion that the *Interaction* within our VR approach is being very well accepted by users and thus gives us a positive answer towards Research Question *RQ-MU-A2*.

#### User Experience

All subjects felt like they were in the same room as their other team member in VR ( $B_1$ ). We therefore conclude that the synchronization of the visualization and representation of the other user works well. The majority of our subjects stated that the VR extension for *ExplorViz* is suitable for team work in their opinion ( $B_2$ ). Only one subject disagreed in this aspect. In our opinion the visualization of the other user, his viewing direction, and the ability to point (with a controller ray) at and highlight objects with the controllers greatly supports team work within our approach. Only one subject recognized signs of motion sickness, with the exception of using the spectating mode) ( $B_3$ ). Thus, we imply that the usage of our VR approach does not cause motion sickness to users in general. Finally most of our subjects (about 82%) would use *ExplorViz* with the VR extension again ( $B_4$ ).

We therefore assume, that our VR approach offers a comfortable virtual environment, which is worth to being applied again. Overall, we received an average agreement rate of about 93% within is category, which states an excellent evaluation of the perceived *User Experience* of our VR approach. This observation allows us to draw the conclusion that applying our VR approach provides a proficient user experience and thus gives us a positive answer towards Research Question *RQ-MU-A1*.

## 9. *ExplorViz* VR Approach Evaluation

### Visual Appearance

All subjects stated, that the structure of the menu is intuitive ( $C_1$ ). Questions  $C_3$  (Visualization of the movement of other users) and  $C_4$  (Number of text insertions) were rated very positive. The visualization of other users ( $C_2$ ), more precisely the other team member, was rated mostly positive. Although, some subjects would have preferred a complete visualized (upper) body of the other user instead of only the HMD and their controllers. We only implemented a simple body visualization of users to offer a larger visual field. Especially if multiple users participate inside a session, we see a major drawback when implementing extended user bodies. Only one subject stated, that text overlays should contain larger text and another one that the text was not readable at the edges of the display ( $C_5$ ). The majority of our subjects was not affected by this circumstances. However, visualized small text labels attached to classes or components were mentioned to be barely readable by a larger share of subjects. One issue depicts still the limited display resolution of the employed HMDs. Thereby, many text labels are very small and a solution for this issue might be to provide a feature to enlarge text labels. Thus, a user could be able to magnify a text, if he points a controller (ray) onto a desirable text label and presses a button. Overall, the *Visual Appearance* category received an average agreement rate of 91%, which depicts a good result. The results allow us to draw the conclusion that the *Visual Appearance* of our VR approach is very well realized and thus gives us a positive answer towards Research Question *RQ-MU-A3*.

### Spectating

Employing the spectating feature within our approach allows a user to see the point of view of a specific user. In exchange, he loses control over his movements in VR. Therefore, we were interested in the perceived user experience and usefulness of this feature. For both questions ( $D_1$  and  $D_2$ ) we received a positive agreement of about 68%, respectively 77%. However, one of four subjects would prefer not to use the spectating feature again. A majority of our subjects stated, that they felt no (motion) sickness while using the spectating mode. Although, seven subjects stated that they felt such sickness. Interestingly, all of these seven subjects were also afraid of

### 9.3. Case Study: Distributed Multi-User VR Environments

heights ( $G_5$ ), so we see a correlation in this case, despite the small sample size. Loosing control in a virtual environment could cause nausea, especially when people are vulnerable to (motion) sickness or afraid of heights. It might be interesting to further investigate this correlation in the future. Unsurprisingly, subjects who felt (motion) sick also stated that they would not use the spectating feature again ( $D_2$ ). A possible explanation might be that the spectating feature was only applied once at the end of the usability study. Thus, the subjects might not have seen reasonable applications of this feature. In our opinion the spectating feature could offer an alternative for users who are not able to physically move within the virtual room on the one hand. On the other hand, we can imagine the feature for demonstration or teaching purposes. Overall, we received an average agreement rate of about 73% within this category. This observation does not allow us to draw the conclusion that spectating mode within our VR approach states a useful feature. Thus, we cannot give a positive answer towards Research Question *RQ-MU-A4*.

#### **Qualitative Feedback**

In addition to our presented results we asked for qualitative feedback by providing a free text field to improve our VR approach. In the following, we present the most frequently stated answers. About 25% of our subjects would welcome a sharper text within the visualization, since reading the text labels despite moving closer was not sufficient in some cases. Tweaking the rendering of the text labels could address this concern. Although we improved the readability based on the evaluation of our previously presented single user VR approach evaluation, the rendering of the text seems to be still an issue. Thus, we need to further improve the rendering of the text within our approach. Another solution might be employing follow-up model HMDs with a higher resolution. A similar share of subjects noted, that the visualization of the other person inside the VR could be enhanced. More precisely, the subjects would prefer a more detailed representation, which would include a body. The visualized model of a user includes only a HMD model and two controller models. We consciously decided to only present a simple model visualization of the users inside the virtual environ-

## 9. *ExplorViz* VR Approach Evaluation

ment to offer a larger visual field. Employing a more detailed or realistic user model would lead in our opinion to a visual reduction. Four subjects suggested to present a more appealing virtual environment within our visualization. The current implementation only offers a very simple environment with a background, colored in light blue, and a textured, carpet-like floor. Some subjects suggested to improve the employed colors and a more beautiful background. One subject even suggested a beach setting. In the future, we plan to offer an alternative environment set, from which the user can choose. The same number of subjects recommended to improve the realization of rotating applications. In order to move an application, it first needs to be bound to the user's controller and then the application can be moved by dragging the controller accordingly. Furthermore, some subjects proposed to rotate an application by moving the analog stick (Oculus Rift). A similar gesture could be realized with the trackpad (HTC Vive). We plan to evaluate, if this movement gesture could be an alternative.

### **Threats to Validity**

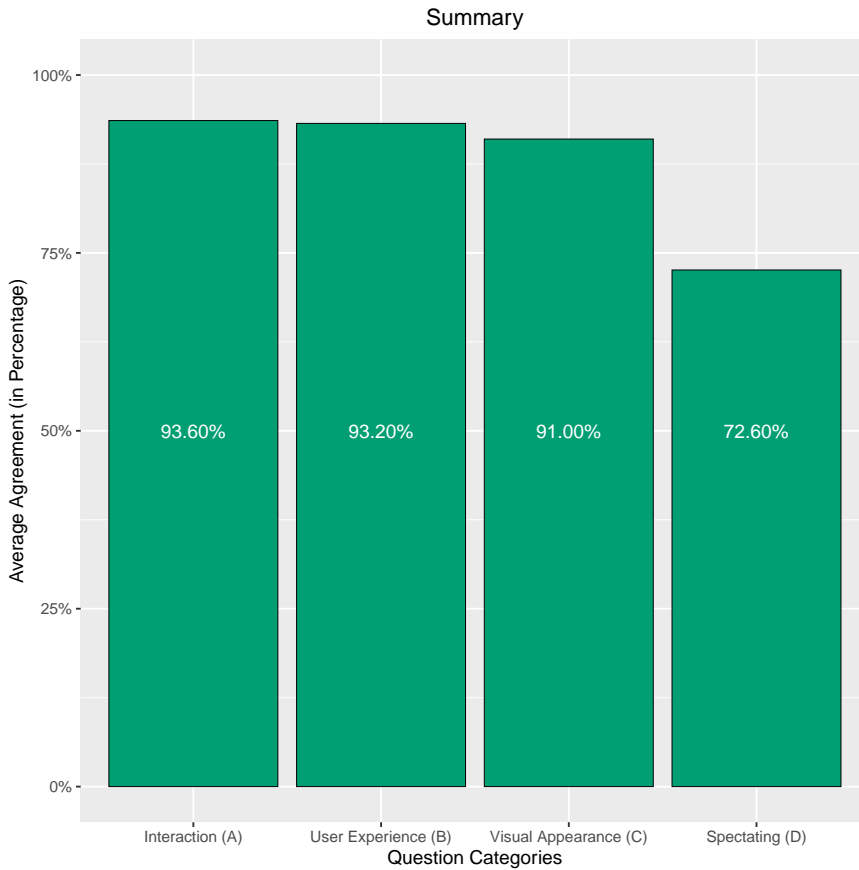
Within our experiment we faced the same threats to validity as in the previous usability study as described in Section 9.2.

### **Summary**

In this evaluation, we presented an usability study to verify the user experience and usability of our collaborative, multi-user VR approach. We found answers to our research questions *RQ-MU-A1*, *RQ-MU-A2*, and *RQ-MU-A3* due to our conducted usability study. All participating teams were able to solve the given tasks, although some needed small hints with respect to the interaction. Regarding Research Question *RQ-MU-A1*, the results allow us to draw the conclusion that applying the interaction inside the virtual environment and controls can be handled well by users and thus offer an intuitive usability. In Figure 9.16, the average agreement rate for our four question categories *Interaction*, *User Experience*, *Visual Appearance*, and *Spectating* are shown. Based on the average agreement rate of 93.2% for the question category *User Experience*, we can confirm Research Ques-



### 9.3. Case Study: Distributed Multi-User VR Environments



**Figure 9.16.** Average agreement rates for the question categories *A - D*.

tion *RQ-MU-A2*, i.e., is the controller-based interaction well accepted by the user? Especially Question  $B_1$  (Users felt like they were together in the virtual room) received an overwhelmingly positive rating. The same applies to Research Question *RQ-MU-A3* ("Are the visual illustrations useful for the user?") with an agreement rate of 91%. Even though the visualized text

## 9. *ExplorViz* VR Approach Evaluation

overlays inside the virtual environment were only moderately well received, the question category *Visual Appearance* was rated very positively overall. However, the visualization of the users inside the virtual environment has potential for improvement based on the feedback of our subjects. Our last research question, *RQ-MU-A4* (“Is the ability to spectate a multi-user VR session useful for the user?”), cannot be confirmed. Thus, we only received an average agreement rate of 72.6% (*Spectating*). An explanation is that we did not clearly stated meaningful use cases for the spectating mode and only applied the mode at the end of our usability study for a single time. This may have left this feature unmotivated and resulted in combination with a potential perceived (motion) sickness of some subjects to this low rating. In upcoming usability studies we plan to take this findings into account.

In particular the question categories *Interaction* and *User Experience* of our VR approach were accepted very well and felt intuitive. The *Visual Appearance* aspect of our approach also received a good evaluation, but leaves room for improvement.

### 9.3.2 Collaborative System and Program Comprehension Evaluation

After we presented a usability evaluation of our multi-user VR approach in Section 9.3.1, we conducted an empirical evaluation focusing on the collaborative applicability of our multi-user VR approach for system and program comprehension in [Brü20]. We describe our experimental design and present the operation, results, discussion, and threats to validity. Finally, we conclude the evaluation by summarizing the results.

#### Experimental Design

We based our evaluation on general software engineering experimentation guidelines [KPP+02; KDJ04; JG12; WRH+12] and conducted an experiment where subjects solved in teams of two program comprehension tasks while applying our VR approach. Once the teams finished the tasks, they were

### 9.3. Case Study: Distributed Multi-User VR Environments

asked to fill out a questionnaire in order to obtain their experience. Furthermore, we (the experiment leaders) also made notes during the experiment to document our observations.

#### Research Questions

We formulate the following four research questions (RQ-MU-B) for the experiment:

- ▷ RQ-MU-B1: Are the newly added features a gain for the use of the VR approach? (Features)?
- ▷ RQ-MU-B2: Is the VR approach applicable for solving complex comprehension tasks in the context of static and dynamic software analysis (Program Comprehension)?
- ▷ RQ-MU-B3: Is the VR approach applicable for solving complex comprehension tasks collaboratively in teams? (Collaborative Program Comprehension)?

In order to verify our research questions, we prepared tasks focusing on collaborative program comprehension applying our multi-user VR approach.

#### Hardware Configuration

For the experiment, the subjects of each team were distributed into two different rooms, referred to as Room A and Room B in the following. Furthermore, we employed three physical machines as shown in Table 9.3. The Oculus Rift (Room B) is bundled with two Oculus Touch controllers (one for each hand) and two Oculus Sensors. The play area is configured to an almost quadratic area of approximately  $4.2 \text{ m}^2$ . The HTC Vive Pro (Room A) uses two Vive Pro controllers and two base stations. The related play area is set to an area of approximately  $14.4 \text{ m}^2$ .

## 9. *ExplorViz* VR Approach Evaluation

**Table 9.3.** Hardware setup employed for the applicability multi-user VR approach evaluation.

	Machine 1	Machine 2	Machine 3
CPU	Intel Core i5-6500		Intel Core i7-6700HQ
GPU	NVIDIA GeForce GTX 1070		NVIDIA GeForce GTX 1060
RAM	16 GB		
HMD	None	Oculus Rift	HTC Vive Pro

### Software Configuration

The machines 1 and 3 use Microsoft Windows in Version 10 Pro - 64 Bit as operating system, Machine 3 uses Microsoft Windows in Version 10 Enterprise - 64 Bit. Both, Machine 2 and 3 run an instance of the *ExplorViz* frontend<sup>12</sup> and have each the *ExplorViz* frontend VR extension<sup>13</sup> installed. Machine 1 runs an instance of the *ExplorViz* backend<sup>14</sup> and *ExplorViz* backend extension VR,<sup>15</sup> to which the other machines connect. Additionally, we installed the ColorPicker<sup>16</sup> extension in Version 1.5.0 to be able to adjust the color theme of the visualization in case one of the subjects is visually impaired. An important aspect constitutes the used browser Firefox in Version 72.0.2, since newer versions do not support the utilized WebVR standard. Finally, we used the built-in headsets of the employed HMDs to allow a communication between the teams and experiment leaders with the help of *Discord*,<sup>17</sup> an instant messaging and VoIP application platform.

For the experiment, we employed the *Common Component Modelling Example (CoCoME)* [HRR16] provided by the iObserve research project [HHJ+13] in Version 1.0<sup>18</sup> from a previous collaboration [HJZ+17; HZJ17]. Basically,

<sup>12</sup><https://github.com/ExplorViz/explorviz-frontend/releases/tag/1.5.0>

<sup>13</sup><https://github.com/ExplorViz/explorviz-frontend-extension-vr/releases/tag/v1.5.0>

<sup>14</sup><https://github.com/ExplorViz/explorviz-frontend/releases/tag/1.5.0>

<sup>15</sup><https://github.com/ExplorViz/explorviz-backend-extension-vr/releases/tag/v1.5.0>

<sup>16</sup><https://github.com/ExplorViz/explorviz-frontend-extension-colorpicker/releases/tag/1.5.0>

<sup>17</sup><https://discord.com>

<sup>18</sup><https://github.com/research-iobserve/cocome-cloud-jee-platform-migration/tree/v1.0-ICSA>

### 9.3. Case Study: Distributed Multi-User VR Environments



**Figure 9.17.** Landscape-level perspective showing the *CoCoME* landscape visualized in *ExplorViz* in Version 1.5.0.

this landscape contains a monitored software landscape representing the open source software *CoCoME* (landscape-level) as shown in Figure 9.17. Furthermore, this landscape also includes an explorable same-named Application *CoCoME* (application level), which can be seen in Figure 9.18.

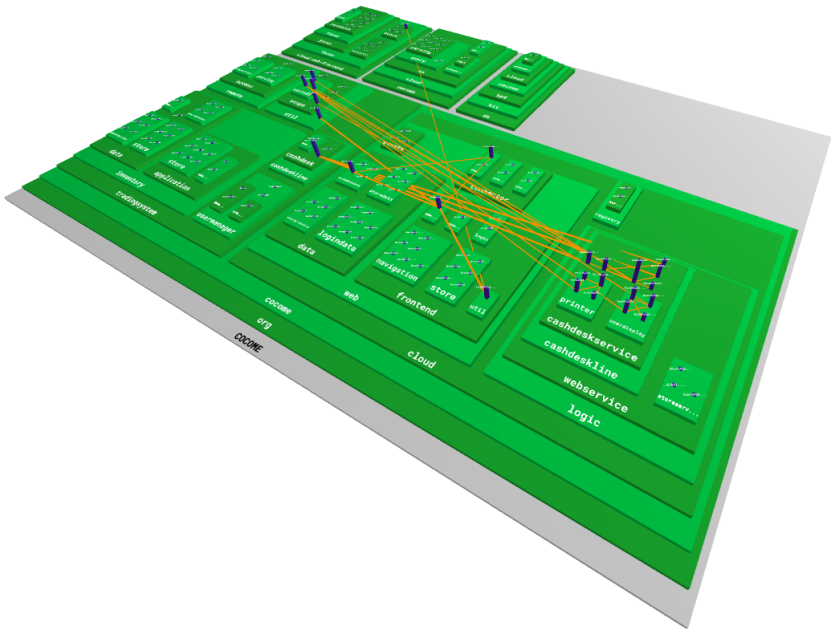
#### Population

For the experiment we were able to motivate 24 subjects and thus 12 teams with a variety of prior VR experience and software visualization to participate. Nine subjects were students, eight were researchers, and seven were working in industry. The average age of our subjects was 32.9 years with a standard deviation of 10.4 years.

#### Operation

In the following, we describe the operation of our experiment. Since evaluating the collaborative applicability of our multi-user VR approach requires more than a single user, the experiment is performed by teams of two within a single session. One subject uses the Oculus Rift and the other one the HTC Vive Pro. The choice of the device rested within the team.

## 9. ExplorViz VR Approach Evaluation



**Figure 9.18.** Application-level perspective showing the CoCoMe application within the same-named landscape visualized in *ExplorViz* in Version 1.5.0.

### Questionnaire

We employed a two-parted, online-based questionnaire built with tool surveyJS.<sup>19</sup> Our motivation was to get rid of paper-based questionnaires on the one hand and to offer two languages (German and English) for our subjects on the other hand. At the beginning of the questionnaire, we provided a short video to give an introduction to *CoCoMe*, which is included in our replication package hosted on *Zenodo*.<sup>20</sup> The first part addressed general information about the subject and was filled out at the beginning of the experiment. Afterwards, we conducted the actual

<sup>19</sup><https://surveyjs.io/>

<sup>20</sup><https://doi.org/10.5281/zenodo.3775572>

### 9.3. Case Study: Distributed Multi-User VR Environments

experiment and the teams performed a sequence of provided tasks focusing on the collaborative applicability of the VR approach. For each task, we noted if their solution was correct or not. Once the subjects finished the tasks they were asked to give some feedback on the VR approach in the second part of the questionnaire. We were especially interested in the perceived user experience, while using the multi-user VR approach, and the applicability for the provided program comprehension tasks. For most of the questions the subjects were asked to answer them based on a 4-Point Likert Scale [Lik32] ranging from 3 (Strongly agree) to 0 (Strongly disagree) without a neutral value. The questionnaire can be found in Appendix A.3. For further information about the experiment we refer to [Brü20].

#### Pilot Study

Before we started the actual experiment, we conducted a small scale pilot study with colleagues as subjects first. Based on the received feedback we improved the assigned tasks and questionnaire.

#### Training Phase

In order to bring all subjects to a similar knowledge level, we conducted a training phase to get them familiar with the HMD, the navigation within the virtual space, and the controller-based interaction. The training took place collaboratively with both subjects of a team within the virtual space and covered the navigation and interaction capabilities step by step. For the training phase, we employed the *Sample Application*<sup>21</sup> provided by *ExplorViz* in combination with *ExplorViz* in Version 1.5.0.<sup>22</sup> Basically, this landscape contains a small, mocked software landscape containing two, explorable instances of the *SampleApplication* (landscape-level) as shown in Figure 9.19. The explorable applications are identical versions of the *SampleApplication* (application level), except for the name of the application, and constitute a minimalist Java application. For load generation, we employed a simple

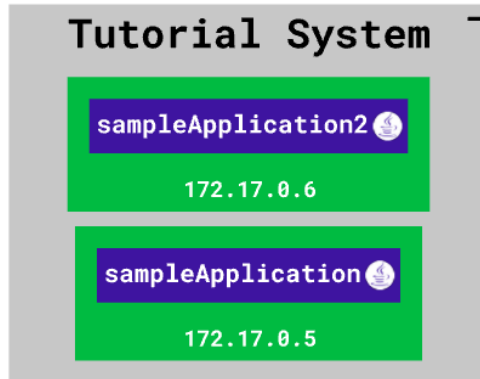
---

<sup>21</sup><https://github.com/ExplorViz/sampleApplication>

<sup>22</sup><https://github.com/ExplorViz/ExplorViz/releases/tag/v1.5.0>

## 9. *ExplorViz* VR Approach Evaluation

load driver, which randomly executes a reconfigured set of Java and JDBC methods Figure 9.20.



**Figure 9.19.** Landscape-level perspective showing two instance of the *SampleApplication* within the tutorial landscape visualized in *ExplorViz* in Version 1.5.0.

### Experiment Phase

After finishing the training phase, the actual experiment began. Since we were interested in the applicability, each team of two subjects was asked to collaboratively solve program comprehension tasks provided by us orally in a specific order.

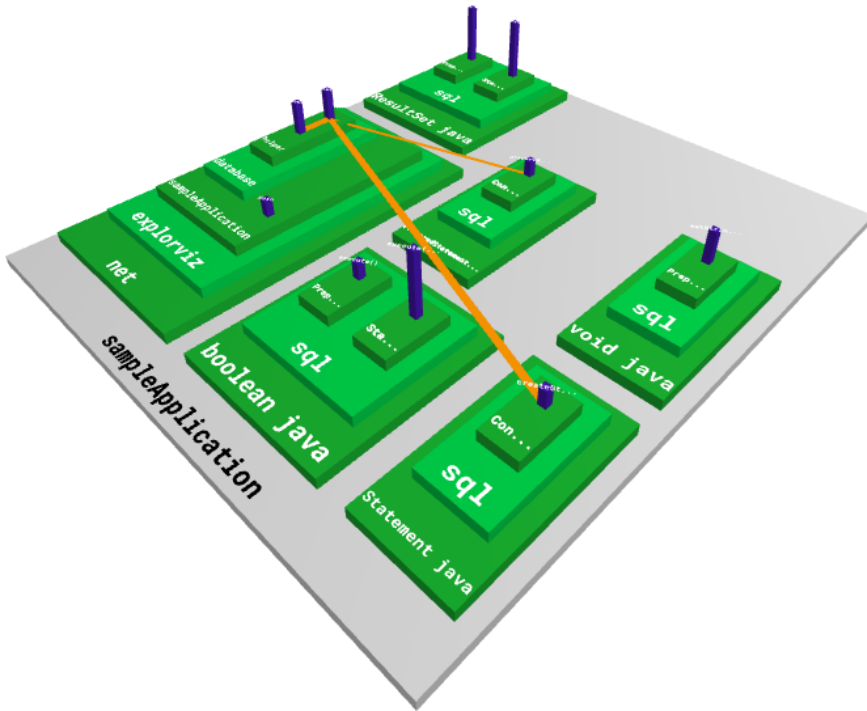
## Results

### Team Relationship

First, we were interested in the relationship of each team. More precisely, we wanted to know how well the team members know each other in advance ( $R_1$ ), from which place they know each other ( $R_2$ ), and how often they have worked as a team before ( $R_3$ ). 29.2% of our subjects know their team member ( $R_1$ ) very well, followed by 37.5% who know them well. Thus,



### 9.3. Case Study: Distributed Multi-User VR Environments



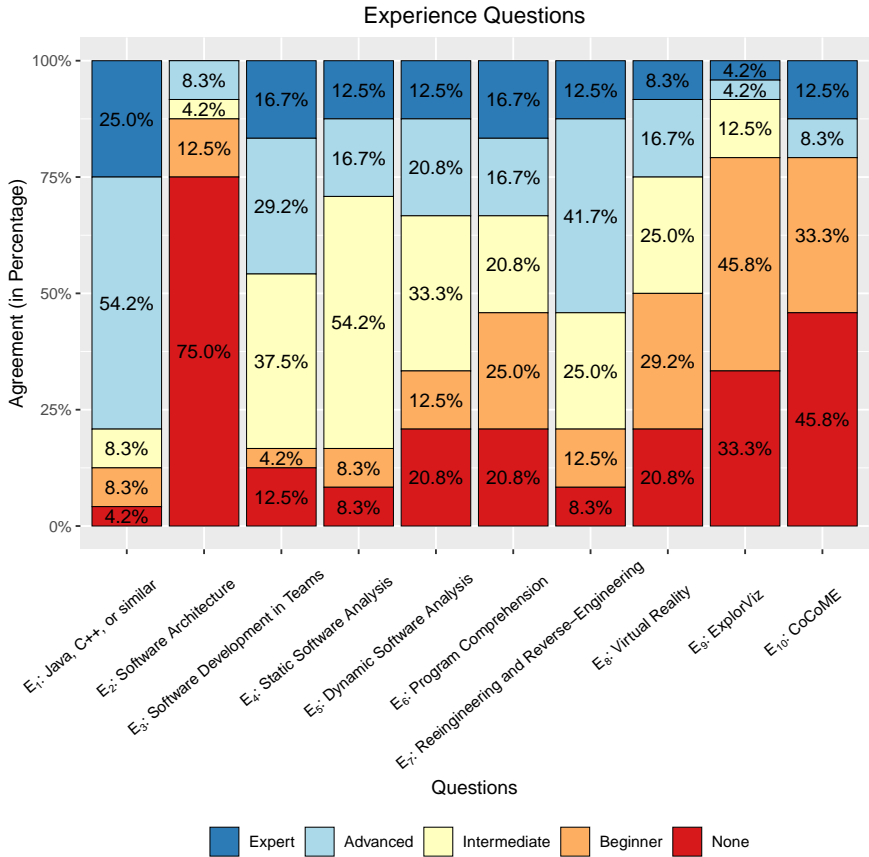
**Figure 9.20.** Application-level perspective showing the first instance of the *SampleApplication* application within the tutorial landscape visualized in *ExplorViz* in Version 1.5.0.

more than two-third of our subjects know their team member at least well. In contrast, only 12.5% stated to know them “Barely”, respectively 20.8% (“Not all all”). Upon the question to specify the origin of their relationship ( $R_2$ ), the subjects stated with the following shares – 25.0% (“None specific”), 58.3% (“From Work”), and 16.7% (“Personally”). Finally, we asked how often they have worked as a team before ( $R_3$ ). 50% of our subjects have worked “often” as a team and 12% at least “sometimes”. The remaining share of 38% stated, that they have “never” worked as a team before.

## 9. ExplorViz VR Approach Evaluation

### Experience Questions

Furthermore, we were interested in prior related experiences, which are related towards our experiment. The results are shown in Figure 9.21. About



**Figure 9.21.** Experiment results for *Experience Questions* ( $E_1$ - $E_{10}$ ).

80% of our subjects stated that they have at least advanced experience in object-oriented programming ( $E_1$ ). In contrast, only about 40% of our

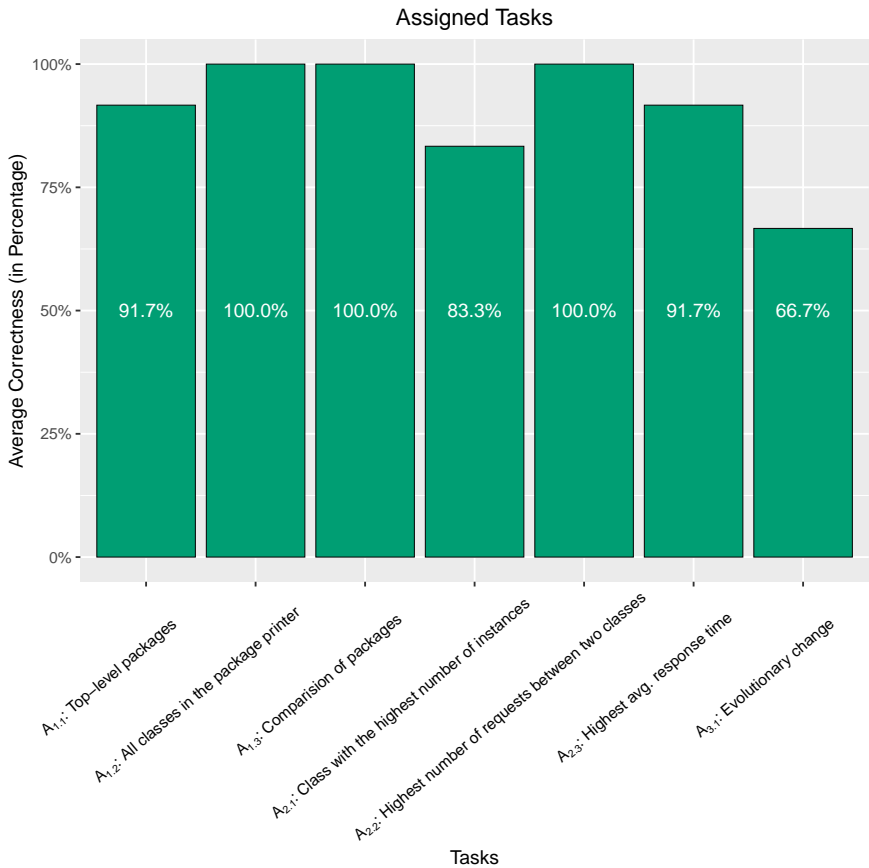
### 9.3. Case Study: Distributed Multi-User VR Environments

subjects have prior experience in software architecture ( $E_2$ ). Question  $E_3$  focuses on the experience of developing software in teams. Here 83.4% of our subjects stated to have intermediate or higher experiences. Only, 12.5% had no prior experience in this aspect. Since our program comprehension tasks built upon the analysis of software, it is relevant to know if our subjects have related experiences. Static software analysis has been employed by 91.7% ( $E_4$ ) and Dynamic Software Analysis ( $E_5$ ) by 79.2% of our subjects. Experience with program comprehension ( $E_6$ ) is nearly balanced between our possible answers. Merely 20.8% of our subjects have no prior experience. As program comprehension is often utilized within reengineering or reverse-engineering tasks, we also want to know about prior knowledge in this aspect ( $E_7$ ). Here, about 80% of our subjects stated to have an experience with the extent of a “beginner” at least. Our next question focuses on the prior experience of a subject with VR ( $E_8$ ). The majority of 79.2% of our subjects has some or even more experience with VR, while about the remaining subjects has no VR experience at all. Our last two questions addressed the prior experience with *ExplorViz* ( $E_9$ ) and the observed software system *CoCoME* ( $E_{10}$ ). While more than two-third of our subjects stated that they have experience with *ExplorViz*, only about the half of our subjects have used *CoCoME* before.

#### Assigned Tasks

Since we were interested in the applicability of our multi-user VR approach, we prepared program-comprehension oriented tasks for our teams. Once a team found a solution for a given task, we noted their answer and decided after the experiment whether their answer is correct or not. The tasks were divided into three categories –  $A_1$ : Structure of the software system ( $A_{1,1}$ - $A_{1,3}$ ),  $A_2$ : System load inspection ( $A_{2,1}$ - $A_{2,3}$ ), and  $A_3$ : Evolutionary change ( $A_{3,1}$ ). The results are shown in Figure 9.22. The average correctness rate is 90.5%. Particular Tasks  $A_{1,2}$ ,  $A_{1,3}$ , and  $A_{2,2}$  were solved correctly by every team. An exception depicts Task  $A_{3,1}$ , which seemed to be a greater challenge with a correctness rate of only 66.7%.

## 9. ExplorViz VR Approach Evaluation

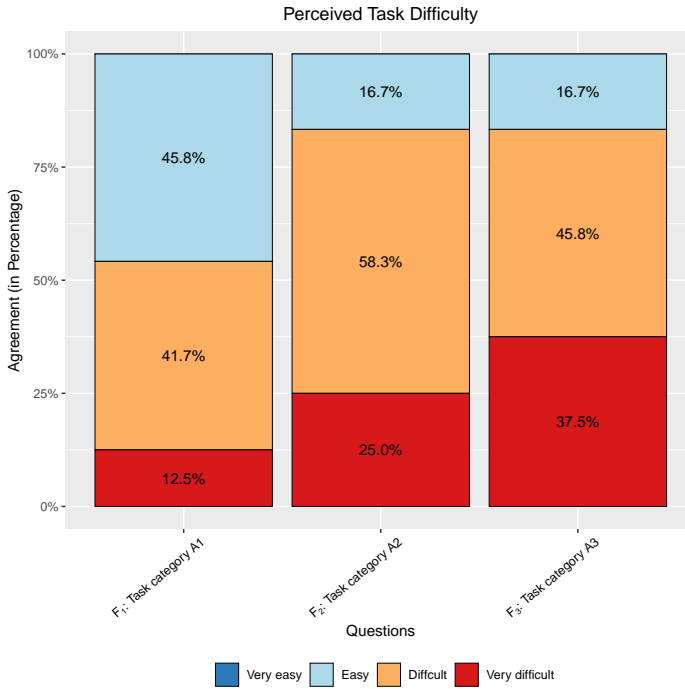


**Figure 9.22.** Experiment results for *Assigned Tasks* ( $A_{1.1}$ - $A_{3.1}$ ).

### Perceived Task Difficulty

Once the subjects finished the experiment, we asked them to give some feedback on the perceived difficulty of the assigned tasks within the experiment first. One can see the increasing perceived difficulty from Task Category  $A_1$  to  $A_3$  within our experiment in Figure 9.23. While Task Category  $A_1$

### 9.3. Case Study: Distributed Multi-User VR Environments



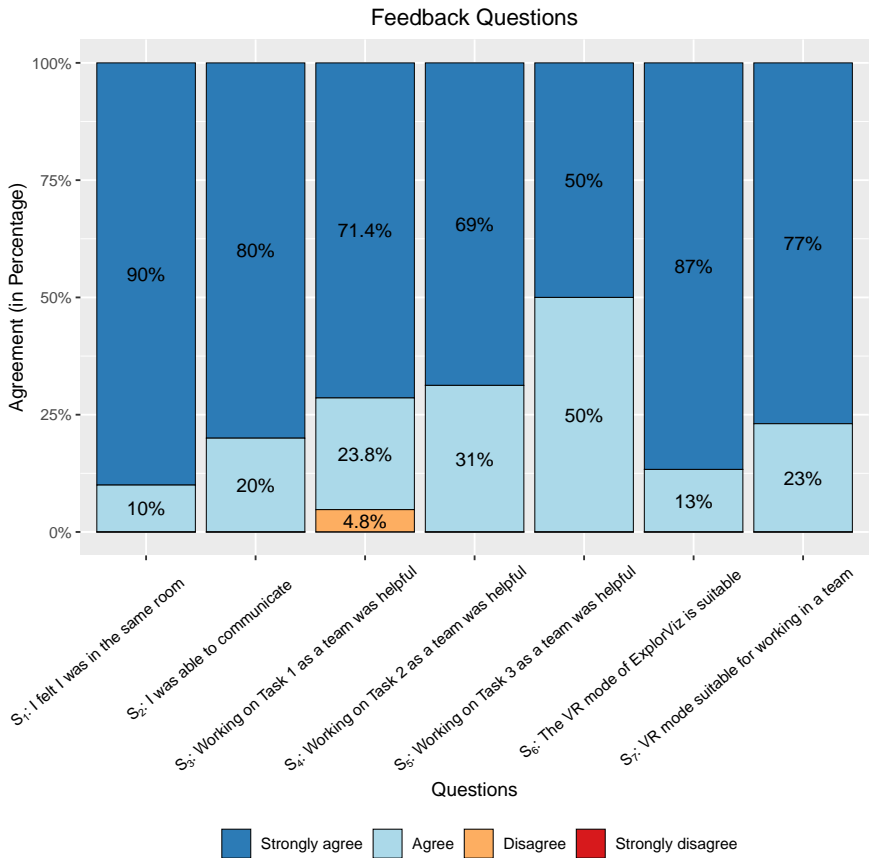
**Figure 9.23.** Experiment results for the *Perceived Task Difficulty* of our task categories ( $A_1 - A_3$ ).

consisted of *easy* tasks for 45.8% of our subjects, the remaining share perceived it as *difficult* or even *very difficult*. Task Category  $A_2$  seemed to be at least *difficult* for 80% of our subjects. We received a similar statement for Task Category  $A_3$ . More precisely, the involved task  $A_{3,1}$  within this task category stated to be a substantial challenge for 37.5% of our subjects.

#### Feedback Questions

Finally, we asked for some feedback regarding the usability and applicability of our VR approach. In Figure 9.24, we present the results for this aspect.

## 9. ExplorViz VR Approach Evaluation



**Figure 9.24.** Experiment results for the *Feedback Questions*.

With an exception to Question  $S_3$ , all subjects agreed upon our presented statements. Only one subject stated that working on Task 1 as a team was not useful for him. Additionally, a majority of our subjects strongly agreed for most statements. All other questions did not receive a single disagreement rating.

## Discussion

In this section we analyze and discuss the results of our previously presented collaborative, multi-user VR experiment. In the following, we present the analysis and discussion of our *Assigned Tasks* and *Feedback Questions*.

### Assigned Tasks

We were especially interested, if our collaborative, multi-user VR approach is applicable for program comprehension tasks. As we mentioned before, the tasks were divided into three categories –  $A_1$ : Structure of the software system ( $A_{1.1}$ - $A_{1.3}$ ),  $A_2$ : System load inspection ( $A_{2.1}$ - $A_{2.3}$ ), and  $A_3$ : Evolutionary change ( $A_{3.1}$ ). The task categories address different use case scenarios and thus system and program comprehension tasks. Task Category  $A_1$  allows to obtain an overview of the observed software system. Task Category  $A_2$  in comparison focuses on identifying bottlenecks in the observed software system based on runtime information. Finally, Task Category  $A_3$  covers potential evolutionary changes of the observed software system. The difficulty of the task categories and included tasks is increasing from Task  $A_1$  to Task  $A_3$ . Identifying the top-level packages within our *CoCoME* application ( $A_{1.1}$ ) received a correctness rate of 91.7%. Some subjects stated, that they had not a clear idea what top-level packages depict within our application-level visualization. One reason might be, that those subjects had only few experiences with object-oriented programming and thus had problems understanding the hierarchical visualization of packages and classes. In contrast, naming all classes within the package printer ( $A_{1.2}$ ) was solved correctly by all teams. Furthermore, the comparison of packages within the visualization ( $A_{1.3}$ ) depicted no difficulty for all teams and was solved correctly in every case. Task  $A_{2.1}$ , naming the class with the highest number of active instances, was correctly solved by 10 teams (about 83% of our subjects). Only two teams were misled by the height of the visualization or completed their search too early and provided an incorrect answer. One solution might be to provide predefined visualization perspectives, which allow to compare the height of the visualized instances (classes) more easily. Identifying the communication (pair) between two classes with the highest number of requests ( $A_{2.2}$ ) in comparison was solved correctly by all teams.

## 9. *ExplorViz* VR Approach Evaluation

Thus, improving the visualization of the communication (e.g., the number of calls) based on the results of previous VR experiments seemed to be successful. A similar task constitutes naming the communication (pair) between two classes with the highest average response time ( $A_{2,3}$ ). All teams, except one, were able to solve this task correctly. Task  $A_{3,1}$ , which focuses on a potential evolutionary change of *CoCoME*, represented a challenge for one third of our teams. Based on the increased difficulty of the task compared to the previous tasks, we did not expect an as high correctness rate as for other tasks. Although, we expected a better result than we received. We asked them to substitute a class with another one and thus name depending classes, which would be affected by the change. One reason might be that the definition of dependencies may have been different among our subjects (i.e., incoming or outgoing dependencies). Providing a definition for dependencies could have made the task more explicit and thus lead to better results.

### Feedback Questions

90% of our subjects felt being in the same room with the other team member in VR ( $S_1$ ). We therefore conclude that the synchronization of the visualization and representation of the other user works well. As our teams were physically separated and working in different rooms, the communication between them and us (experiment leaders) was a crucial part of the experiment. 80% of our subjects stated that they could communicate and interact with their team member well. As we were particularly interested in the collaborative applicability of our approach, we asked the subjects if working as a team on the Task Categories  $A_1 - A_3$  was helpful ( $S_3 - S_5$ ). Although one subject did not agree on this aspect for Task Category  $A_1$ , all other subjects stated that working on a team was helpful for them. This confirms our conviction that there is a need for collaborative approaches in the area of system and program comprehension. Based on the presented results so far, our collaborative, multi-user VR approach provides a solution for this. The majority of our subjects (87%) stated that the VR approach for *ExplorViz* is suitable for team work ( $S_6$ ). In our opinion the visualization of the other user, his viewing direction and the ability to point (with a controller ray)



### 9.3. Case Study: Distributed Multi-User VR Environments

at and highlight objects with the controllers greatly supports team work within our approach. Finally, most of our subjects (77.7%) stated that the VR approach is suitable for working in teams ( $S_7$ ). We therefore assume, that our VR approach offers a collaborative and productive virtual working environment, which is worth to being applied again. Overall, we received an agreement for every question within is category, except for Question  $S_3$  (one subject), which states an excellent feedback of our VR approach.

#### Qualitative Feedback

In addition to our presented results we asked for qualitative feedback to improve our VR approach. In the following, we present the most frequently given answers. A large number of subjects noted, that the capability to highlight entities within the visualization is a useful feature. Although selecting specific communication lines was cumbersome. Providing the capability to magnify an entity in order to select it more easily might be a solution. Based on the previous evaluations of our VR approach we improved the readability of visualized text labels. Nevertheless, some subjects had still issues to read some text labels, especially when long text labels were shortened. Thus, we need to further improve the rendering of the text within our approach. Another mentioned aspect was the performance upon opening or closing packages within the visualization. Due to the employed, large *CoCoME* software landscape, the delay upon performing these actions led sometimes to a slight latency. Currently, we are evaluating different approaches for improving the rendering and layouting to decrease the existing latency. One subject suggested to represent the direction of the communication in the visualization in addition to the employed information box attached to the controller. Additionally, an option to preserve the displayed information on some kind of virtual white board would be useful, especially if it would be collaboratively usable. This suggestions could improve the usability and thus could be realized in the future. The last remarks focus on the execution of the experiment. Our interactive VR tutorial was stated as very helpful by multiple subjects, notably the controller mapping. Also providing and explaining the assigned tasks to the teams was perceived in a similar way. Finally, the communication of the

## 9. *ExplorViz* VR Approach Evaluation

team and us (experiment leaders) via *Discord* was disturbed in some cases based on an echo. This was caused when two persons (experiment leader and subject) were speaking at the same time in a room (Room A or Room B). This aspect is an artificial boundary condition based on the experiment setup and should not encounter during an ordinary use of the approach.

### Threats to Validity

Within our experiment we faced the same threats to validity as in the previous usability study as described in Section 9.2. Differences are described in the following.

**Training Phase** In contrast to the previous usability study, where we provided simple navigation and interaction tasks, we asked the subjects to perform complex, program-comprehension oriented tasks within our experiment. The chosen tasks might be not adequate in design or scope to comprehensively verify the applicability and usability of our collaborative, multi-user VR approach.

**Tasks** Furthermore, we noticed during the experiment, that some subjects had problems to remember the controller mapping for navigating and interacting in the visualization like in the previous usability studies. We provided a solution for this aspect by adding an option to display the mapping during the experiment inside the visualization. Unfortunately, some subjects had trouble to get into the related menu.

### Summary

In this evaluation, we presented an experiment to verify the user experience and applicability of our collaborative, multi-user VR approach. For replication and archiving purposes we published the experiment and the related setup and results on *Zenodo*.<sup>23</sup> We found answers to our research questions *RQ-MU-B1*, *RQ-MU-B2*, and *RQ-MU-B3* due to our conducted experiment.

---

<sup>23</sup><https://doi.org/10.5281/zenodo.3775572>

### 9.3. Case Study: Distributed Multi-User VR Environments

All participating teams were able to solve the majority of the assigned tasks, although some teams did not correctly solve every one. Regarding Research Question *RQ-MU-B1* ("Are the newly added features a gain for the use of the VR approach (Features)?"), the results allow us to draw the conclusion that working on the evaluations of our previous VR experiments and implementing missing features was a success. The newly integrated, visualized detail boxes have increased the amount of information which can be obtained from a chosen entity (e.g., a communication line). Furthermore, the capability to display the actual controller assignment stated a valuable added feature. However, selecting single communication lines within the experiment was difficult for some teams, so this issue should be addressed in the future. The high correctness rates, especially for Task Categories  $A_1$  and  $A_2$  indicate that our VR approach is applicable for program comprehension tasks. In combination with the high average agreement of Feedback Question  $S_6$ , we can draw the conclusion that the VR approach is suitable for static and dynamic analysis tasks. Based on this results, we can confirm Research Question *RQ-MU-B2*, i.e., is the VR approach applicable for solving complex comprehension tasks in the context of static and dynamic software analysis (Program Comprehension)? Our last research question *RQ-MU-B3* investigates, if the VR approach is applicable for solving complex comprehension tasks collaboratively in teams (Collaborative Program Comprehension). Again, the high success rates and the received feedback for the different task categories indicate that the collaborative usage of our VR approach is helpful. The members of the teams knew each other in the average case and had worked together at least occasionally before. This context may have contributed to a successful cooperation and thus high correctness rate. Although three teams were not familiar with each other before, two of the three teams solved all tasks correctly. Thus, we conclude that the familiarity within a team is not a mandatory aspect for an efficient collaborative usage of our VR approach for solving program comprehension tasks in a team. Overall, the results allow us to draw the conclusion, that our multi-user VR approach is applicable for solving complex program comprehension tasks in teams.

## 9. *ExplorViz* VR Approach Evaluation

### 9.4 Summary

In this chapter, we evaluated the usability and applicability of our VR approach *ExplorViz* VR for collaborative program comprehension by conducting three, consecutive experiments.

#### Single-User VR Approach - Usability

First, we evaluated the initial version of our single-user VR approach focusing on usability aspects. In particular the *Navigation* of our VR approach was accepted very well and felt intuitive for the subjects. The *User Experience* and *Interaction* aspects of our approach also received good evaluations, but leaves room for improvement. In combination with our observations we imply that our VR approach combines the advantages of 3D visualizations with navigation and interaction capabilities in a virtual environment.

#### Collaborative Multi-User VR Approach - Usability

In our second evaluation, we conducted an experiment to verify the user experience and usability of our collaborative, multi-user VR approach. Again, the *Interaction* and *User Experience* of our improved VR approach were accepted very well and felt intuitive. The *Visual Appearance* aspect of our approach also received a good evaluation, but leaves room for improvement. Although our *Spectating* proved not to be a useful feature, we imply that our VR approach offers an intuitive and practical handling for the presented 3D software visualization. The experiment gave a first, successful impression of the collaborative capability of our multi-user VR approach.

#### Collaborative Multi-User VR Approach - Applicability

Finally, we performed a third experiment to investigate the applicability of our multi-user VR approach for program comprehension tasks, especially in teams. Based on the results of the experiment, we conclude that our multi-user VR extension is applicable for collaborative program comprehension

#### 9.4. Summary

tasks in the context of static and dynamic software analysis. We learned from our previous experiments and implemented a list of useful features, which enhanced the usability and applicability of our approach. Minor visualization issues regarding the visibility of the communication lines depending on the status of opened or closed components misled some subjects and thus resulted in incorrect answers. As a consequence, this aspect needs to be addressed in an upcoming version.



# *CORAL* Approach Evaluation

In this chapter, we present a qualitative evaluation of our *CORAL* approach. First we define the goals for our evaluation in Section 10.1. Afterwards, we describe the application of *CORAL* with three successful iterations in form of a case study involving the open source research project *ExplorViz* in Section 10.2. Finally, in Section 10.3, we summarize the results of the presented evaluation.

## 10. CORAL Approach Evaluation

### Previous Publications

Parts of this chapter are already published in the following works:

1. Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, and Dan Kröger. “Microservice Decomposition via Static and Dynamic Analysis of the Monolith.” In: *Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2020
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
3. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
4. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49



## 10.1 Goals

For the evaluation, we focus on two essential aspects of our *CORAL* approach, namely modularization and reengineering on the one hand, and collaborative software engineering on the other hand. Again, we follow the GQM approach [BW84; SB99; SBC+02] to define our evaluation goals, which are explained in the following.

### **CORAL-G1: Modularization and Reengineering**

Our first evaluation goal is to verify, if our *CORAL* approach is applicable for software systems and related projects within the goal to perform a modernization and modularization process. More precisely, we want to investigate, if our consecutive, iterative approach can be applied to a software system multiple times until a sufficient state is reached.

### **CORAL-G2: Collaborative Software Engineering**

Our second last evaluation goal is an extension of our previous goal. Modularizing and reengineering a software system is one of the most important aspects of our *CORAL* approach. Furthermore, we are also interested, if our approach is also able to improve an observed software system in a collaborative manner while our approach is applied. Here, we focus on the software itself and the related development process.

## 10.2 Case Study: *ExplorViz*

In this section, we evaluate the *CORAL* approach by applying it to the open source research project *ExplorViz* moving from a monolithic towards a microservice architecture as a proof-of-concept and case study. The new architecture facilitates a collaborative development process for both researchers and students. We explain how we employed our iterative modularization and reengineering approach *CORAL* applied measures, and describe how we solved occurring issues and enhanced our development

## 10. CORAL Approach Evaluation

process. Afterwards, we illustrate the application of our modularization approach and present the modernized, extensible software system architecture and highlight the improved collaborative development process. After the first iteration of the process, we present a proof-of-concept implementation featuring several developed extensions in terms of architecture and extensibility. After conducting the second iteration, we achieved a first version of a microservice architecture and an improved development process with room for improvement, especially regarding service decoupling. Finally, as a result of the third iteration, we illustrate our improved implementation and development process representing an entire, separately deployable, microservice architecture. In order to make the different versions through the iterations visible, we later call the outdated version *ExplorViz Legacy*, and the new version just *ExplorViz*.

The remainder of this evaluation is organized as follows. In Section 10.2.1, we illustrate our problems and drivers for a modularization and architectural modernization. Afterwards, we present the initial state our software system and underlying architecture of *ExplorViz Legacy* in Section 10.2.2. The following first iteration of this process as well as the target architecture of *ExplorViz* are described in Section 10.2.3. In Section 10.2.4, we conclude the first iteration with a proof-of-concept implementation in detail, including an evaluation based on several developed extensions. The second iteration of our process in terms of achieving a first microservice architecture is presented in Section 10.2.5. As there was still room for improvement, we describe how we further improved our microservice architecture and development process in Section 10.2.6. Finally, the conclusions are drawn and an outlook is given.

### 10.2.1 Problem Statement

The open source research project *ExplorViz* started in 2012 as part of a PhD thesis and is further developed and maintained until today. *ExplorViz* enables a live monitoring and visualization of large software landscapes [Flo16; FRH15]. In particular, the tool offers two types of visualizations – a landscape-level and an application-level perspective. The first provides an overview of

## 10.2. Case Study: *ExplorViz*

a monitored software landscape consisting of several servers, applications, and communication in-between. The second perspective visualizes a single application within the software landscape and reveals its underlying architecture, e.g., the package hierarchy in Java, and shows classes and related communication. The tool has the objective to aid the process of system and program comprehension for developers and operators. We successfully employed the software in several collaboration projects [HJZ+17; HZJ17] and experiments [FKH15b; FFH+15].

The project is developed from the beginning on *Github* with a small set of core developers and many collaborators (more than 40 students) over the time. Several extensions have been implemented since the first version, which enhanced the tool's feature set. Unfortunately, this led to an unstructured architecture due to an unsuitable collaboration and integration process. In combination with technical debt and issues of our employed software framework and underlying architecture, we had to perform a technical and process-oriented modularization. Since 2012, several researchers, student assistants, and a total of 31 student theses as well as multiple projects contributed to *ExplorViz*. We initially chose the Java-based Google Web Toolkit framework (*GWT*) [Ope20b], which seemed to be a good fit in 2012, since Java is the most used language in our lectures. *GWT* provides different wrappers for Hypertext Markup Language (HTML) and compiles a set of Java classes to JavaScript (JS) to enable the execution of applications in web browsers. Employing *GWT* in our project resulted in a monolithic application (hereinafter referred to as *ExplorViz Legacy*), which introduced certain problems over the course of time.

### 10.2.1.1 Extensibility & Integrability

*ExplorViz Legacy's* concerns are divided in core logic (core), e.g., predefined software visualizations, and extensions. When *ExplorViz Legacy* was developed, students created new Git branches to implement their given task, e.g., a new feature. However, there was no extension mechanism that allowed the integration of features without rupturing the core's code base. Therefore, most students created different, but necessary features in varying classes for the same functionality. Furthermore, completely new technologies were uti-

## 10. CORAL Approach Evaluation

lized, which introduced new, sometimes even unnecessary (due to the lack of knowledge), dependencies. Eventually, most of the developed features could not be easily integrated into the master branch and thus remained isolated in their created feature branch.

### 10.2.1.2 Code Quality & Comprehensibility

After a short period of time, modern JS web frameworks became increasingly mature. Therefore, we started to use *GWT*'s JavaScript Native Interface (JSNI) to embed JS functionality in client-related Java methods. For example, this approach allowed us to introduce a more accessible JS-based rendering engine. Unfortunately, JSNI was overused and the result was a partitioning of the code base. Developers were now starting to write Java source code, only to access JS, HTML, and Cascading Style Sheets (CSS). This partitioning reduced the accessibility for new developers. Furthermore, the integration of modern JS libraries to improve the user experience in the frontend was problematic. Additionally, Google announced that JSNI would be removed with the upcoming release of Version 3 (*GWT*), which required the migration of a majority of client-related code. Google also released a new web development programming language, named *DART*, which seemed to be the unofficial successor of *GWT*. Thus, we identified a potential risk, if we would perform a version update. Eventually, JSNI reduced our code quality. Our remaining Java classes further suffered from ignoring some of the most common Java conventions and resulting bugs. Students of our university know and use supporting software for code quality, e.g., static analysis tools such as *Checkstyle* [Ope20a] or *PMD* [Ope20e]. However, we did not define a common code style supported by these tools in *ExplorViz Legacy*. Therefore, a vast amount of extensions required a lot of refactoring, especially when we planned to integrate a feature into the core.

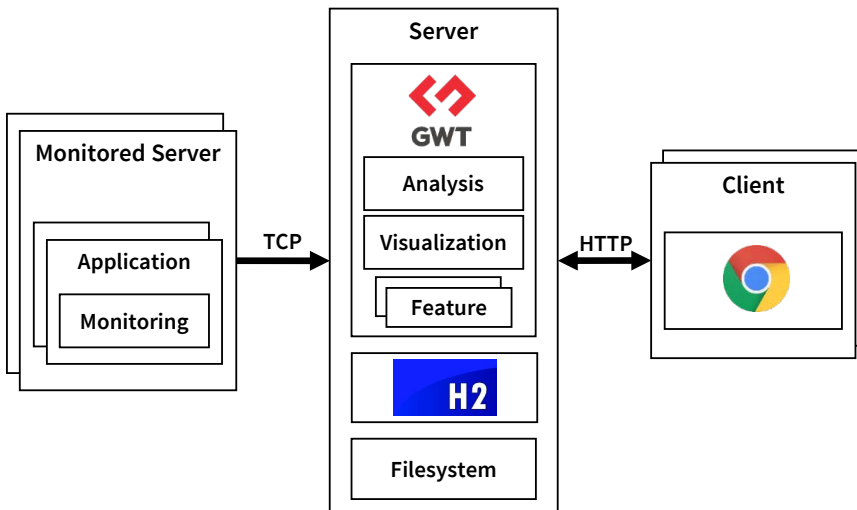
### 10.2.1.3 Software Configuration & Delivery

In *ExplorViz Legacy*, integrated features were deeply coupled with the core and could not be easily taken out. Often, users did not need all features, but only a certain subset of the overall functionality. Therefore, we introduced

new branches with different configurations for several use cases, e.g., a live demo. Afterwards, users could download resulting artifacts, but the maintenance of related branches was cumbersome. Summarized, the stated problems worsened the extensibility, maintainability, and comprehension for developers of our software. Therefore, we were in need of modularizing and modernizing *ExplorViz Legacy*.

### 10.2.2 *ExplorViz Legacy*

To understand the modularization process, we provide more detailed information about our old architecture in the following.



**Figure 10.1.** Architectural overview and employed software stack of the monolithic *ExplorViz Legacy*.

In Figure 10.1, the overall architecture and the employed software stack of *ExplorViz Legacy* is shown. We are instrumenting applications, regardless whether they are native applications or deployed artifacts in an application

## 10. CORAL Approach Evaluation

server like *Apache Tomcat*.<sup>1</sup> The instrumentation is realized by our monitoring component, which employs in the case of Java *AspectJ*, an aspect-oriented programming extension for Java [Ecl20]. *AspectJ* allows us to intercept an application by bytecode-weaving in order to gather necessary monitoring information for analysis and visualization purposes. Subsequently, this information is transported via Transmission Control Protocol (TCP) towards a server, which hosts our *GWT* application. This part represents the two major components of our architecture, namely *analysis* and *visualization*. The *analysis* component receives the monitoring information and reconstructs traces. These traces are stored in the file system and describe a software landscape consisting of monitored applications and communication in-between. Our user-management employs a *H2* database [Ope20c] to store related data. The software landscape *visualization* is provided via Hypertext Transfer Protocol (HTTP) and is accessible by clients with a web browser. *GWT* is an open source framework, which allows to develop JS front-end applications in Java. It facilitates the usage of Java code for server (backend) and client (frontend) logic in a single web project. Client-related components are compiled to respective JS code. The communication between frontend and backend is handled through asynchronous remote procedure calls (ARPC) based on HTTP. The usage of ARPC allows non-professional developers, in our case computer science students, to easily extend our existing open source research project. ARPC enables a simple exchange of Java objects between client and server. In *ExplorViz Legacy*, the advantages of *GWT* proved to be a drawback, because every change affects the whole project due to its single code base. New developed features were hard-wired into the software system. Thus, a feature could not be maintained, extended, or replaced by another component with reasonable effort. This situation was a leading motivation for us to look for an up-to-date framework replacement. We intended to take advantage of this situation and modularize our software system to move from a monolithic to a distributed (web) application divided into separately maintainable and deployable backend and frontend components.

Our open source research project is publicly accessible since the begin-

---

<sup>1</sup><http://tomcat.apache.org>

## 10.2. Case Study: *ExplorViz*

ning on *GitHub* and is licensed under the *Apache License, Version 2.0*. Our development process facilitated the maintainability and extensibility of our software by means of so-called feature branches. Every code change, e.g., a new feature or bugfix, whether it was developed by a researcher, a student assistant, or a student during a thesis or project, had to be implemented in a separated feature branch based on the master branch. After performing a validation on the viability and quality of the newly written source code, the branch had to be merged into the master branch and thus permanently into the project. This fact often led to an intricate and time-consuming integration process, since all developers worked on a single code base. For that reason, we had to improve our development process to perform a modularization and technical modernization.

The previously mentioned drawbacks in *ExplorViz Legacy* and recent experience reports in literature about successful applications of alternative technologies, e.g., Representational State Transfer (REST or RESTful) Application Programming Interfaces (API) [UZX+11; Vin08], were triggers for a modularization and modernization.

### 10.2.3 First Iteration: Modularization Process and Architecture of *ExplorViz*

Within *ExplorViz Legacy*, we applied the *CORAL* approach, which guided us through our modularization process from performing a first requirement analysis and defining goals towards our actual state. Summarized, we performed multiple iterations of the process until we reached an entire, maintainable, and particularly extensible microservice architecture. In the following, the first iteration of the process is described.

#### 10.2.3.1 Requirement Analysis and Goals

We no longer perceived advantages of preferring *GWT* over other web frameworks. During the modularization planning phase, we started with a requirements analysis for our modernized software system and identified technical and development process related impediments in the project. We kept in mind that our focus was to provide a collaborative development

## 10. CORAL Approach Evaluation

process, which encourages developers to participate in our open source research project [ZKH18]. Furthermore, developers, especially inexperienced ones, tend to have potential biases during the development of software, e.g., they make decisions on their existing knowledge instead of exploring unknown solutions [TRP+17].

As a result, we intended to provide plug-in mechanisms for the extension of the backend and frontend with well-defined interfaces. We intended to encourage developers to try out new libraries and technologies, without rupturing existing code. According to [LBG+15], the organization of a software system implementation is not an adequate representation of a system's architecture. Thus, architectural changes towards the implementation of a software system have to be documented before or at least shortly after the realization. If this aspect is not addressed, the architecture model has at least to be updated based on the implementation in a timely manner. Thus, we took this into account to enhance our development process. Architectural decay in long-living software systems is also an important aspect. Over time, architectural smells manifest themselves into a system's implementation, whether they were introduced into the system from the beginning or later during development [LLS+18]. For the modularization process of our software system it was necessary to look for such smells to eliminate them in the new system. In the end, we identified the following goals for our modularization and modernization process:

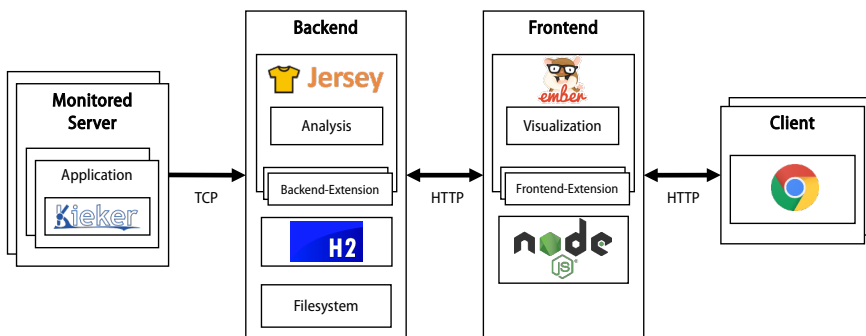
- ▷ The project needs to be stripped down to its core, anything else is a form of extension.
- ▷ We need to focus on the main purpose of our project – the visualization of software landscapes and architectures. Thus, we need to look for a monitoring alternative.
- ▷ The backend and frontend should be separately deployable and technologically independent. The latter goal allows us to exchange them with little effort. Additionally, they store their own data and use no centralized storage or database.
- ▷ Scaffolds or dummy-projects should be provided for the development of



extensions.

- ▷ We stick to the encapsulation principle and provide well-defined interfaces.
- ▷ The overall development process needs to be enhanced, e.g, by using Continuous Integration (CI) and quality assurance, like code quality checks.

In general, there exist many drivers and barriers for microservice adoption [KH19]. Typical barriers and challenges are the required additional governance of distributed, networked systems, and the decentralized persistence of data.



**Figure 10.2.** Architectural overview and software stack of the modularized *ExplorViz* (after the first iteration).

After we applied the two actions *Manual Analysis* and *Tools-based Analysis* within our iterative *CORAL* approach, we agreed within the *Recommendation* action to build our recommendation plan upon an architecture based on microservices. This architectural style offers the ability to divide monolithic applications into small, lightweight, and independent services, which are also separately deployable [KH18; HS17; DGL+17; AAE16]. However, the obtained benefits of a microservice architecture can bring along some drawbacks, such as increased overall complexity and data consistency [CBD18].

## 10. CORAL Approach Evaluation

Adopting the above mentioned goals led us finally to the microservice-based architecture shown in Figure 10.2.

### 10.2.3.2 Extensibility & Integrability

In a first step, we modularized our *GWT* project into two separated projects, i.e., backend and frontend, which are now two self-contained microservices. Thus, they can be developed technologically independent and deployed on different server nodes. In detail, we employ distinct technology stacks with independent data storage. This allows us to exchange the microservices, as long as we take our specified APIs into account. The backend is implemented as a Java-based web service based on *Jersey* [Ora20], which provides a RESTful API via HTTP for clients. *Jersey* implements the Servlet 3.0 specification, which offers `javax.servlet.annotations` to define servlet declarations and mappings. We assume that the usage of the Servlet 3.0 specification eases the development process in the backend, especially for students. Furthermore, we replaced our custom-made monitoring component by the monitoring framework *Kieker* [HKG+11; HWH12; HH20]. This framework provides an extensible approach for monitoring and analyzing the runtime behavior of distributed software systems. Monitored information is sent via TCP to our backend, which employs the filesystem and *H2* database for storage. *Kieker* employs a similar monitoring data structure, which fits our replacement requirements perfectly. The frontend uses the JS framework *Ember.js*, which enables us to offer visualizations of software landscapes to clients with a web browser [Emb20]. Since *Ember.js* is based on the *model-view-viewmodel* architectural pattern, developers do not need to manually access the Document Object Model and thus need to write less source code. *Ember.js* uses *Node.js* as execution environment and emphasizes the use of components in web sites, i.e., self-contained, reusable, and exchangeable user interface fragments [Joy20]. We build upon these components to encapsulate distinct visualization modes, especially for extensions. Communication, like a request of a software landscape from the backend, is abstracted by so-called *Ember.js* adapters. These adapters make it easy to request or send data by using the *convention-over-configuration* pattern. The introduced microservices, namely backend and frontend, represent the

core of *ExplorViz*. As for future extensions, we implemented well-defined extension interfaces for both microservices that allow their integration into the core.

### 10.2.3.3 Code Quality & Comprehensibility

New project developers, e.g., students, do not have to understand the complete project from the beginning. They can now extend the core by implementing new mechanics on the basis of a plug-in extension. Extensions can access the core functionality only by a well-defined read-only API, which is implemented by the backend, respectively frontend. This high level of encapsulation and modularization allows us to improve the project, while not breaking extension support. Additionally, we do no longer have a conglomeration between backend and frontend source code, especially the mix of Java and JS, in single components. This eased the development process and thus reduced the number of bugs, which previously occurred in *ExplorViz Legacy*. Another simplification was the use of *json:api* [Ope20d] as data exchange format specification between backend and frontend, which introduced a well-defined JavaScript Object Notation (JSON) format with attributes and relations for data objects. This minimizes the amount of data and round trips needed when making API calls. Due to its well-defined structure and relationship handling, developers are greatly supported when exchanging data.

### 10.2.3.4 Software Configuration & Delivery

One of our goals was the ability to easily exchange the microservices. We fulfill this task by employing frameworks, which are exchangeable with respect to their language domain, i.e., Java and JS. We anticipate that substituting these frameworks could be done with reasonable effort, if necessary. Furthermore, we offer pre-configured artifacts of our software for several use cases by employing *Docker* images. Thus, we are able to provide containers for the backend and frontend or special purposes, e.g., a fully functional live demo. Additionally, we implemented the capability to plug-in developed extensions in the backend, by providing a package-scanning

## 10. CORAL Approach Evaluation

mechanism. The mechanism scans a specific folder for compiled extensions and integrates them at runtime.

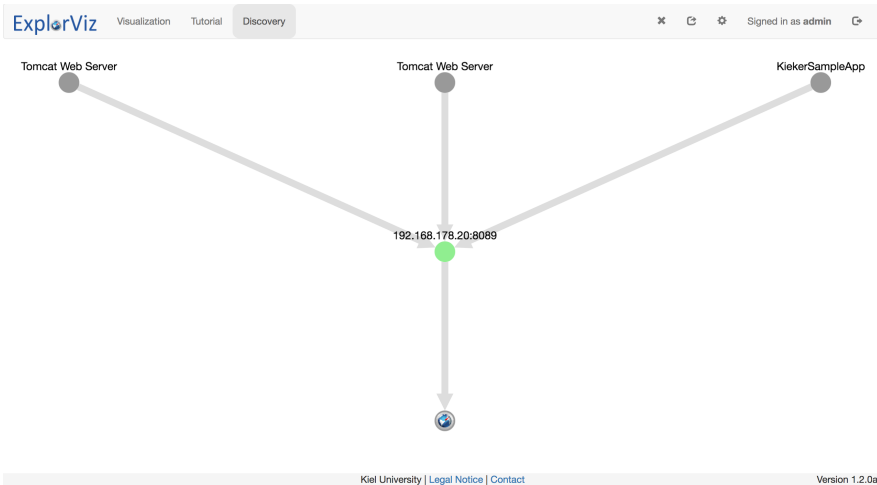
### 10.2.4 Proof-of-Concept Implementation

To execute and afterwards evaluate the recommendation plan we designed before, we realized a proof-of-concept implementation and split our project as planned into two separate projects – a backend project based on *Jersey*, and a frontend project employing the JS framework *Ember.js*. Both frameworks have a large and active community and offer sufficient documentation, which is important for new developers. As shown in Figure 10.2, we strive for an easily maintainable, extensible, and plug-in-oriented microservice architecture. Since the end of the first iteration of our modularization and modernization process in early 2018, we were able to successfully develop several extensions both for the backend and the frontend. Four of them are described in the following.

#### 10.2.4.1 Application Discovery

Although we employ the monitoring framework *Kieker*, it lacks a user-friendly, automated setup configuration due to its framework characteristics. Thus, users of *ExplorViz* experienced problems with instrumenting their applications for monitoring. In [KZH18], we reported on our application discovery and monitoring management system to circumvent this drawback. The key concept is to utilize a software agent that simplifies the discovery of running applications within operating systems. An example visualization of the extension's user-interface is shown in Figure 10.3. The figure shows three discovered applications on a monitored server. Furthermore, this extension properly configures and manages the monitoring framework *Kieker*. More precisely, the extension is divided in a frontend extension, providing a configuration interface for the user, and a backend extension, which applies this configuration to the respective software agent lying on a monitored server. Then, the software agent is able to apply the chosen configuration towards *Kieker* for the application monitoring.

## 10.2. Case Study: *ExplorViz*



**Figure 10.3.** Screenshot of the application discovery extension of *ExplorViz* showing three detected applications on a monitored server.

Finally, we were able to conduct a first pilot study to evaluate the usability of our approach with respect to an easy-to-use application monitoring. The improvement regarding the usability of the monitoring procedure of this extension was a great success. Thus, we recommend this extension for every user of *ExplorViz*.

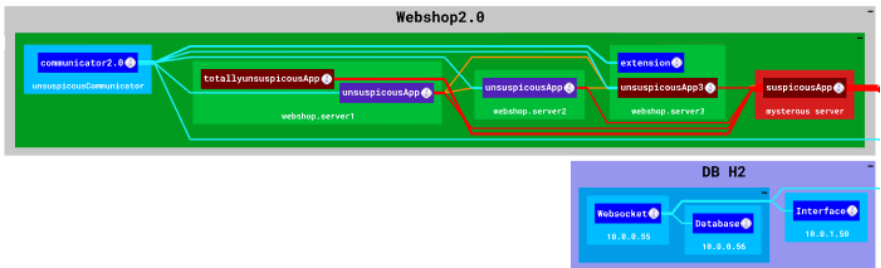
### 10.2.4.2 Virtual Reality Support

An established way to understand the complexity of a software system is to employ visualizations of software landscapes. However, with the help of visualization alone, exploring an unknown software system is still a potentially challenging and time-consuming task. For this extension, five students followed a new approach using Virtual Reality (VR) for exploring software landscapes collaboratively based on our previous work [ZKH19a] as described in Section 9.2 and Section 9.3. They employed head mounted displays (HTC Vive, HTC Vive Pro, and Oculus Rift) to allow a collaborative exploration and comprehension of software in VR. The collaborative VR

## 10. CORAL Approach Evaluation

approach builds upon our microservice architecture and employed Web-Socket connections to exchange data to achieve modular extensibility and high performance for this real-time multi-user environment.

### 10.2.4.3 Architecture Conformance Checking



**Figure 10.4.** Screenshot of the architecture conformance checking (ACC) extension of *ExplorViz* showing two checked systems.

Software landscapes evolve over the time, and consequently, architecture erosion occurs. This erosion causes high maintenance and operation costs. Thus, performing architecture conformance checking (ACC) is an important task. ACC allows faster functionality changes and eases the adaptation to new challenges or requirements. Additionally, software architects can use ACC to verify a developed version against a previous modeled version. This can be used to check whether the current architecture complies with the specified architecture and can reveal constraint violations. An example architecture conformance visualization of a monitored software landscape against a modeled one is shown in Figure 10.4. The visualization illustrates missing or modified (colored in red), and additional (colored in blue) nodes and applications and related communication in-between for a software landscape. In this extension, a student developed an approach to perform an ACC between a modeled software landscape consisting of applications using an editor and a monitored software landscape [Hac18]. Thus, enabling a visual comparison between both versions on an architectural level. To evaluate the extension, the student conducted a usability study with five

participants, applying the model editor for a desired software landscape and performing an ACC of a modeled software landscape against a monitored one. The results indicated a good user experience of the approach, although the usability of the editor could be improved.

### 10.2.4.4 Visualizing Architecture Comparison

Identifying architectural changes between two visualizations of a complex software application is a challenging task, which can be supported by appropriate tooling. Although *ExplorViz* visualizes the behavior and thus the runtime architecture of a software system, it is not possible to easily compare two versions. In this extension one student developed an approach to perform a visual software architecture comparison of two monitored applications, e.g., indicating removed or changed components or classes [Weg18]. This facilitates a developer to see at a glance which parts of the architecture have been added, deleted, modified, or remained unchanged between the two versions. Finally, an evaluation based on a qualitative usability study with an industrial partner was conducted. Five professional software engineers participated in the study and solved comparison tasks based on two different versions of their own developed software. The evaluation showed that the extension is applicable for solving architecture comprehension tasks with different versions within *ExplorViz*.

### 10.2.5 Second Iteration: Restructured Architecture and new Process

As we were not satisfied with the results of the first iteration and some drawbacks remained, we decided to perform a second iteration of our modularization and modernization approach. Our modularization approach started by dividing the old monolith into separated frontend and backend projects [ZKH18]. Since then, we further decomposed our backend into several microservices to address the problems stated in Section 10.2.1. The resulting, restructured architecture is illustrated in Figure 10.5 and the new collaborative development process is described below. As reported in Section 10.2.4, the new architecture already improved the collaboration

## 10. CORAL Approach Evaluation

with new developers who realized new features as modular extensions.

### 10.2.5.1 Extensibility & Integrability

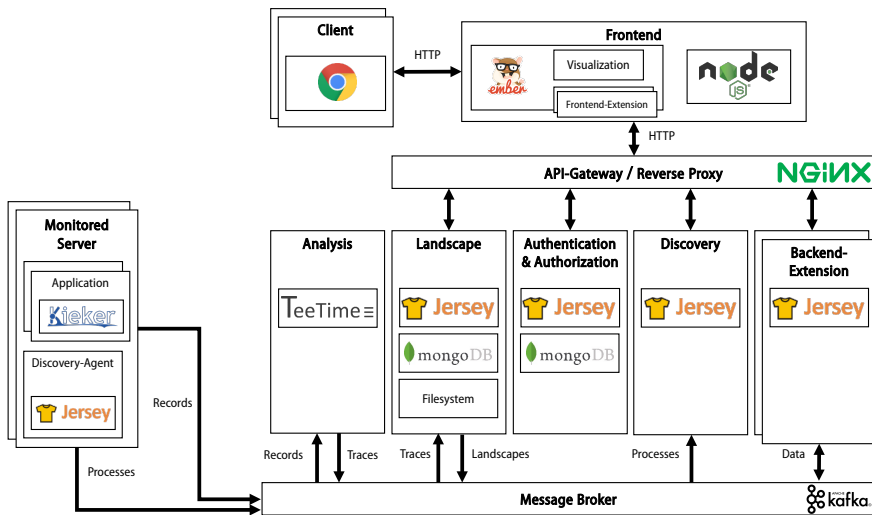
Frontend extensions are based on *Ember.js*'s addon mechanism. This approach works quite well for us as shown in Section 10.2.4. The backend, however, used the package scanning feature of *Jersey* to include extensions. The result of this procedure was again an unhandy configuration of a monolithic application with a high coupling of its modules. Therefore, we once again restructured the approach for our backend plug-in extensions. The extensions are now decoupled and represent separated microservices. As a result, each extension is responsible for its own data persistence and error handling. Due to the decomposition of the backend, we are left with multiple Uniform Resource Identifiers (URI). Furthermore, new extensions will introduce additional endpoints and therefore more URIs again. To simplify the data exchange handling based on those endpoints, we employ a common approach for microservice-based backends. The frontend communicates with an API gateway instead of several single servers, thus only a single base Uniform Resource Locator (URL) with well-defined, multiple URIs. This gateway, a *NGINX* reverse proxy [NGI20], passes requests based on their URI to the respective proxied microservices, e.g., the *landscape-service*. Furthermore, the gateway acts as a single interface for extensions and offers additional features like caching and load balancing. Extension developers, who require a backend component, extend the gateway's configuration file, such that their frontend extension can access their complement. Some extensions must read data from different services. In the past, we used HTTP requests to periodically obtain this data. Each request was processed by the providing service, therefore introducing unnecessary load. The inter-service communication is now realized with the help of *Apache Kafka* [Apa20]. *Kafka* is a distributed streaming platform with fault-tolerance for loosely coupled systems. We use *Kafka* for events that might be interesting for upcoming microservices. For example, the *landscape-service* consumes traces from the respective *Kafka* topic and produces a new landscape every tenth second for another topic. Microservices can consume the topic, obtain, and process the data in their custom way. As a result, the producing service does not have



## 10.2. Case Study: *ExplorViz*

to process unnecessary HTTP requests, but simply fires its data and forgets it. Simple Create Read Update Delete (CRUD) operations on resources, e.g., users and their management, are provided by means of RESTful APIs by the respective microservices. The decomposition into several independent microservices and the new inter-service communication approach both facilitate low coupling in our system.

### 10.2.5.2 Code Quality & Comprehensibility



**Figure 10.5.** Architectural overview and software stack of *ExplorViz* (after the second iteration).

The improvements for code quality and accessibility, which were introduced in the first iteration of our modularization approach, showed a perceptible impact on contributor's work. For example, recurring students approved the easier access to *ExplorViz* and especially the obligatory exchange format *json:api*. However, we still lacked a common code style in terms of conventions and best practices. To achieve this and therefore

## 10. CORAL Approach Evaluation

facilitate maintainability, we defined compulsory rule sets for the quality assurance tools *Checkstyle* and *PMD*. In addition to *SpotBugs* [Ope20g], we impose their usage on contributors for Java code. For JS, we employ *ESLint* [ESL20], i.e., a static analysis linter, with an *Ember.js* community-driven rule set. The latter contains best practices for *Ember* applications and rules to prevent programming flaws. In the future, we are going to enhance this rule set with our custom guidelines. Another aspect are CI tools. CI systems and tools are used to automate the compilation, building, and testing of software (systems). Software projects that employ CI, release twice as often, accept pull requests faster, and have developers who are less worried about breaking the build, compared to projects that do not use CI [HTH+16]. Therefore, employing CI tools is a good method to improve our development process even more. Consequently, we integrated the previously mentioned tools into our continuous integration pipeline configured in *TravisCI* [Ope20j]. More precisely, we employ *TravisCI* for *ExplorViz*'s core and any extension to build, test, and examine the code. Integrating the quality assurance tools allows us to define thresholds within the pipeline. If a threshold regarding quality assurance problems is exceeded, the respective *TravisCI* build will fail and the contributor is notified by email. A similar build is started for each pull request that we receive on *Github* for the now protected master branch. Therefore, contributors are forced to create a new branch or fork *ExplorViz* to implement their enhancement or bug fix and eventually submit a pull request.

### 10.2.5.3 Software Configuration & Delivery

One major problem of *ExplorViz Legacy* was the necessary provision of software configurations for different use cases. The first iteration of modularization did not entirely solve this problem. The backend introduced a first approach for an integration of extensions, but their delivery was cumbersome. Due to the tight coupling at source code level we had to provide the compiled Java files of all extensions for download. Users had to copy these files to a specific folder in their already deployed *ExplorViz* backend. Therefore, configuration alterations were troublesome. With the architecture depicted in Figure 10.5 we can now provide a jar file for each

## 10.2. Case Study: *ExplorViz*

service with an embedded web server. This modern approach for Java web applications facilitates the delivery and configuration of *ExplorViz*'s backend components. In the future, we are going to ship ready-to-use *Docker* images for each part of our software. The build of these images will be integrated into our CI pipeline. Users are then able to employ *Docker Compose* files to achieve their custom *ExplorViz* configuration or modify a provided *Docker Compose* file to fit their needs. As a result, we can provide an alternative, easy to use, and exchangeable configuration approach that essentially only requires a single command line instruction. The frontend requires another approach, since (to the best of our knowledge) it is not possible to install an *Ember.js* add-on inside of a deployed *Ember.js* application. We are currently developing a build service for users that ships ready-to-use, pre-built configurations of our frontend. Users can then download and deploy these pre-built packages. Alternatively, these configurations will also be usable as *Docker* containers.

### 10.2.5.4 Software Architecture Erosion & Accessibility

One of our initial problems was the partitioning of our code base and the resulting software architecture erosion. We think that both employed frameworks, *Ember.js* and *Jersey*, matter when it comes to this problem. *Ember.js* is well documented and there are many examples on how to solve a problem with the framework. Due to its JavaScript nature, we can easily introduce and use modern features in web development. Furthermore, the framework *Ember.js* introduces recognizable and reusable structures which facilitate the development. For the *Jersey* backend, we again provide a sample project that contributors can use for starters. The project is runnable and shows how to use *Kafka* and the HTTP client for different needs. *ExplorViz* uses the monitoring framework *Kieker* to obtain monitoring data. These so called *records* are then processed by the analysis component of our software. The setup of *Kieker* is extensive, but also quite complex for untrained users. Since we are dealing with many students, we were in need of a solution to circumvent this drawback. We developed an external component with a frontend and backend extension that simplifies the monitoring setup for users. The so called discovery agent searches for running Java processes

## 10. CORAL Approach Evaluation

in the encompassing operating system and sends its data to the related discovery backend extension. The frontend discovery extension visualizes the gathered data and provides Graphical User Interface (GUI) forms for users to start and stop the monitoring of found processes (see Section 10.2.4). Ultimately, the resulting discovery mode was successful in internal tests and we integrated it as core feature.

### 10.2.6 Third Iteration: Achieving an entire Microservice Architecture

The second iteration of our modernization process introduced multiple microservices for different backend logic. For example, each backend extension was built as a separate source code project and deployed as a Java jar file. This introduced advantages, among others, for the configuration of *ExplorViz* as described in Section 10.2.5. Since then, we further refined our microservice decomposition. The current architecture, after performing a third iteration of our modularization approach, is illustrated in Figure 10.6. Additionally, we revised the ubiquitous problems as we did in the previous iterations.

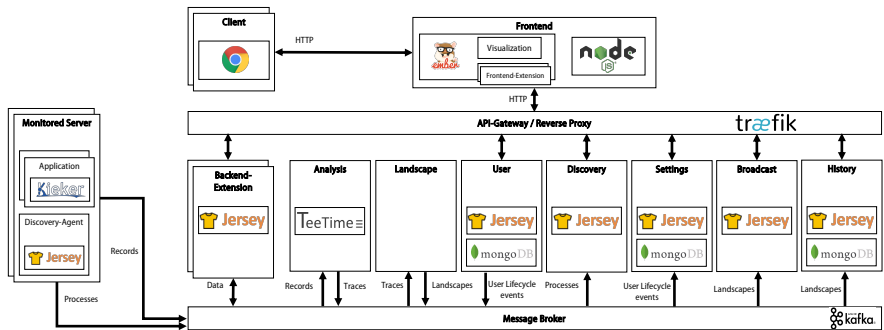


Figure 10.6. Current architectural overview and software stack of *ExplorViz* (after the third iteration).

### 10.2.6.1 Extensibility & Integrability

Both previous iterations shared the problem that collaborators had to implement their feature or extension against the latest version of *ExplorViz*. To circumvent this drawback, we now push the backend build artifacts of the *TravisCI* build pipeline as snapshots to *Sonatype* [Ope20f], i.e., an online maven repository for unsigned artifacts. Furthermore, we use *Github* releases to version *ExplorViz*. These releases follow a documented release management process. As a result, release descriptions and names share a common theme. In general, *Github* releases use Git tags to reference the specific Git commit that represents the release. We use these resulting Git tags for versioning. The tags are picked up by our CI pipeline and are used to name the *Sonatype* snapshots. As a result, contributors can now select specific (intermediate) versions to implement against.

After employing the second iteration of our modernization for some time with different configurations, we observed performance issues regarding the *landscape-service*. This service continuously built our hierarchical landscape model, provided the latest snapshot of the model per HTTP API, and returned previous snapshots upon incoming HTTP requests. We identified that we could decompose these functionalities into separated microservices to distribute the load on one hand and gain a better performance on the other hand. The decoupling of the *landscape-service* can be seen in Figure 10.6. Frontend extensions now register at the *broadcast-service* to receive server-sent events (SSE) which contain the latest landscape model snapshot. Furthermore, specific snapshots can be requested at the *history-service*. This microservice is responsible for storing landscape model snapshots.

### 10.2.6.2 Code Quality & Comprehensibility

Introducing static analysis tools to our CI pipeline showed improvements of *ExplorViz*'s code style. The automatic CI build for *Github* pull requests highlights flaws and allows us to impose refactoring before merging the code. This is also used for collaborators' extensions. Now, the remaining part to improve the overall code quality was testing the source code and the integration of components. We observed that collaborators had less

## 10. CORAL Approach Evaluation

problems with testing frontend extensions than with testing the related backend project. We think that is due to the *Ember.js* documentation and the huge number of already existing open source projects, which already show how one can comprehensively test *Ember.js* projects. Therefore, we wrote sample unit, integration, and API tests for our microservices, which students can use as foundation to test their own written code. By choosing these three categories of tests, we now cover testing at source code and API level. All these tests are automatically executed as part of our CI pipeline. If a test requires other running services, e.g., the reverse proxy, these services are (if necessary) built and executed by means of a *Docker* container.

To further ease the development for collaborators, we wrote supplemental guides on best practices, design ideas, and specifications. These can be found in our public *Github* documentation wiki [Exp20]. Furthermore, our CI pipeline now automatically builds the latest API documentation (*JavaDoc* for the backend and *YUIDoc* for the frontend). The resulting websites are deployed by means of *Github* pages, i.e., public websites based on the content of Git repositories. We additionally employ *Swagger* [Ope20h], an interactive API development editor and UI, to document our HTTP APIs. The tool is automatically executed when a microservice is started in development mode.

### 10.2.6.3 Software Configuration & Delivery

*ExplorViz* enables users and developers to use extensions on demand by providing the build artifacts for every (release) version. We now facilitate *ExplorViz*' configuration with the help of *Docker* images. After pushing the build artifacts to *Sonatype* in the CI pipeline, we subsequently build a *Docker* image for each service and push it to *Docker Hub*. Therefore, users and collaborators can use the publicly hosted *Docker* images to easily create their custom deployment environment with *Docker*.

We build upon this process and now provide ready-to-use *Docker Compose* files for release versions of *ExplorViz*. These configurations allow users to start the core features of *ExplorViz* with only a single command. This approach is also used in the development phase. Since *ExplorViz* requires

auxiliary software, i.e., database management systems (e.g., MongoDB),<sup>2</sup> *Apache Kafka*, and the reverse proxy *NGINX*, we now provide a *Docker Compose* file to start the mandatory, already configured software stack for development. As a result, collaborators do not need to read different instructions on how to start specific software, but only need to start a set of *Docker* containers with the help of the *Docker Compose* file.

Figure 10.6 shows that we replaced our employed reverse proxy *NGINX* with *Traefik* [Ope20i]. The reverse proxy *NGINX* uses a static configuration file to define its routing. As a result, *ExplorViz* users needed to update this configuration or use a provided version to enable an installed or developed extension. This was quite cumbersome and potentially deterred users to try out extensions. With *Traefik* we can now use labels, i.e., metadata for *Docker* objects, to define the routing at *Docker Compose* level. Therefore, the routing of the reverse proxy can be easily extended or changed.

### 10.2.7 Conclusion and Future Work

In this case study, we reported on our modularization and modernization process of the open source research software *ExplorViz*, moving from a monolithic architecture towards a microservice architecture with the primary goal to ease the collaborative development, especially with students. We described technical and development process related drawbacks of our initial project state until 2016 in *ExplorViz Legacy* and illustrated our modularization process and architecture. The process included not only a decomposition of our web-based application into several components, but also technical modernization of applied frameworks and libraries. Driven by the goal to easily extend our project in the future and facilitate a contribution by inexperienced collaborators, we offer a plug-in extension mechanism for our core project, both for backend and frontend. On the basis of *ExplorViz Legacy*, we employed our iterative, collaborative modularization and reengineering process *CORAL* as a guidance through our modularization and performed three successful iterations to *ExplorViz Legacy* until we reached a sufficient state.

---

<sup>2</sup><https://www.mongodb.com>

## 10. CORAL Approach Evaluation

After our first iteration, we realized our modularization process and architecture in terms of a proof-of-concept implementation and evaluated it afterwards by the development of several extensions of *ExplorViz*. Each of these extensions was developed by students and evaluated afterwards, in each case by at least a usability study. The results showed an overall good usability of each extension. In the case of our developed application discovery extension, we integrated it into our core project based on the high-quality of the extension in addition to the good usability and time saving aspect when instrumenting applications with *Kieker*. As the results of the modularization process were not sufficient yet, we performed a second iteration featuring a first microservice architecture. Furthermore, the iteration led to several independent deployable services bundled with inter-service communication handled via the message broker *Kafka* and requests from the frontend towards the backend are passed through our reverse-proxy in form of *NGINX*. Furthermore, we enhanced our development and build process towards a more collaborative manner. Unfortunately, we were not satisfied with the results of the second iteration, because some services were still very large and poorly maintainable. Thus, we needed to perform a further decoupling of them. Additionally, we recognized that our release management and CI processes, as well as our documentation, still needed to be improved. Consequently, with these drawbacks in mind, we performed a third iteration, after which we achieved a fully decoupled microservice architecture, consisting of a set of self-contained systems and well-defined interfaces in-between. The inter-service communication is still handled through *Kafka*. Additionally, we switched our reverse-proxy towards *Traefik* for handling requests from the frontend towards the backend. For the release management and documentation, we further optimized our CI pipeline regarding *Docker* images and supplemental (API) documentation for developers and users. In the future, we are planning to evaluate our finalized project, especially in terms of developer collaboration. Additionally, we plan to move from our CI pipeline towards a continuous delivery (CD) environment. Thus, we expect to further decrease the interval between two releases and allow users to try out new versions, even development snapshots, as soon as possible. Furthermore, we plan to use architecture recovery tools like [GCF+17] for refactoring or documentation purposes in



upcoming versions of *ExplorViz*.

### 10.2.8 CORAL Approach Applicability Assessment

While we applied the *CORAL* approach multiple times on *ExplorViz* we demonstrated its collaborative modularization and modernization capabilities on a real open source software system and project. Although it served us as a comprehensive guidance through our modularization and modernization process moving from a monolithic towards a microservice architecture, we identified some areas of improvement within the approach. Thus, we will list a few aspects, which should be addressed in the future. First, applying a software architecture evaluation method within the manual analysis action proved to be more delicate than we expected, because we had no (external) stakeholders of *ExplorViz*. This circumstance made it difficult for us to apply the method straightforward and could have distorted the process. Thus, we have to take this into account in the future. Another aspect embodies the evaluation action of the approach. Despite we performed a visual software architecture and software metrics comparison after each iteration, it was tedious to evaluate the performed changes due to missing baselines. In the future, baselines should be defined upfront the iterations to ease the process. Finally, our *CORAL* approach and its application on *ExplorViz* should be assessed in a more methodological way based on defined quality criteria and metrics.

## 10.3 Summary

In this chapter, we presented a case study for applying our *CORAL* approach on the open source research software *ExplorViz* over a time span of three years. We successfully performed three iterations of our approach while moving *ExplorViz* from a monolithic towards a more easily extensible and maintainable architecture based on microservices. Furthermore, we achieved a plugin-architecture capable to deal with backend and frontend related concerns, which facilitates the development of extensions, especially for inexperienced collaborators. Additionally, we were able to improve

## 10. *CORAL* Approach Evaluation

the development process, involving activities like documentation, testing, building, and shipping in combination with Continuous Integration (CI). Thus, we were able to verify the applicability for an established open source research project and software system.

# Related Work

In this chapter, we discuss related work regarding our presented *CORAL* approach and involved visualizations. For a general overview on software visualization and software visualization evaluation we refer to [MIK+16; MGA+18; CMZ+20]. We start with presenting related approaches focusing on program comprehension based on 3D software visualization in Section 11.1. Afterwards, we describe closely related work towards our collaborative, multi-user VR approach in Section 11.2. Then, in Section 11.3 we discuss related work on approaches focusing on the visualization of databases. Next, we present related work of approaches, which focus on the modernization of monolithic applications towards a microservice architecture in Section 11.4. Finally, we describe in Section 11.5 which collaboration concepts and tools are related towards our work.

## 11. Related Work

### Previous Publications

Parts of this chapter are already published in the following works:

1. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
3. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
4. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49

## 11.1 Program Comprehension based on 3D Software Visualization

Our multi-user VR approach, as described in Section 7.3, offers system and program comprehension of observed software systems based on the city metaphor presented by Wettel and Lanza [WL07]. For an overview regarding 3D software visualization, we refer to [MZ15]. In the following, we present related work on approaches and tools that also support comprehension tasks based on this kind of 3D software visualization.

SynchroVis [WWF+13] visualizes monitoring traces in 3D with the city metaphor for analyzing concurrency. The tool presents the static structure of an observed software system and provides a detailed visualization of concurrent behavior. Instead of visualizing concurrent behavior of an observed application, we focus on the structure and communication within a software landscape. Kobayashi et al. [KKY+13] visualize components and layers of software systems using a city metaphor in combination with a generated map in their tool SARF Map. The tool visualizes implicit software features, which are represented as a city block and related classes of a feature are shown as buildings reflecting their software layer. By this way, developers and stakeholders can utilize the tool collaboratively for high-level discussions and thus make decisions for the future development. Instead of a combined visualization with a generated map with the aim to ease the process of decision-making and communicating, we rely only on the city metaphor and focus on system and program comprehension for developers and operators alike. Chronotwigger [ERS+14] enables the visualization of source and test files in a visual analytics system with interlinked 2D and 3D display views of mined Git repositories. The multi-user and collaborative software visualization tool supports program comprehension based on source code and test code in addition to co-evolution. Furthermore, users are able to select a specific time span and select nodes with the visualization and perform zooming actions to analyze co-changes. To offer a more immersive user experience the 3D visualization is displayed on a wall display. In contrast to Chronotwigger, we do not aim to visualize and

## 11. Related Work

thus improve the software development process but instead target system and program comprehension.

Weninger et al. [WMM19; WMM20] propose an approach to visualize the evolution of utilized memory over time using the city metaphor. Thus, they are able to reveal the dynamic memory behavior of an observed application. Since the visualization is continuously updated the user gets the impression of an evolving city. By using the approach, users can identify and inspect heap objects and properties like threads to comprehend memory usage and changes. In contrast, our approach reveals the structure and behavior of a software system. More precisely, we focus on supporting system and program comprehension for developers and operators. Benomar et al. [BSP13] propose a framework which offers a visualization environment based on heat maps. In detail, they aim to support program comprehension while exploring time and other dimensions of a software system. In comparison to their approach, we employ a visualization to comprehend the structure and dynamic behavior of a software system but do not facilitate heat maps. BlendedCity [DMM+15] is an approach where a mixture of colors is utilized to reveal different aspects of a visualized software system. Thus, in addition to the structural information depicted by the city metaphor the authors integrate information about source code changes, stack traces, and IDE interaction information in the visualization. The visualization considers activity data, but maps this information on represented static entities. In comparison to BlendedCity, our approach is limited to dynamic analysis data based on monitoring information. Thus, we only visualize the structure of the software system and the communication in-between and are not capable to present stack traces or IDE interaction.

### 11.2 Virtual Reality

At the time we started with the first version of our single-user VR approach, as described in Section 7.2, only a small set of related approaches existed. Meanwhile, a larger amount of innovative approaches using software visualization bundled with VR focusing on program comprehension tasks came

along. A selection of closely related approaches and tools is presented in the following.

CityVR [MGA+17] and VRCity [VNP17] are both interactive software visualization tools that use the city metaphor in combination with VR to improve the process of program comprehension. The approaches facilitate physical movements within their respective virtual environments for navigation and focus on object-oriented software. VRCity was also extended with an AR (Augmented Reality) capability in addition to the VR variant in [SNB+19]. In contrast, our approach also offers the capability to move physically, but also allows for teleportation within the virtual environment. Furthermore, we are not limited to object-oriented software systems, because our flexible monitoring based on *Kieker* allows us to instrument a large set of different programming languages and frameworks. In [MSF+18; MSZ+18] the authors present *IslandViz*, an approach for exploring OSGi-based software systems in VR. OSGi (Open Services Gateway Initiative) depicts a dynamic module-based framework specification for Java. The approach is based on an island metaphor to emphasize the modular aspects of OSGi and offers a visualization of the related software system laying on a virtual table in front of the user. Instead of an island metaphor, we employ the city metaphor as our visualization technique and focus on the structure and behavior of the observed software system. Romano et al. present their tool *Code2City* and the related VR-based implementation *Code2CityVR* in [RCE+19b]. Within their approach the user utilizes a controller and a HMD to immerse himself in the virtual environment. The visualization shows static analysis information for selected entities, e.g., the lines of code or the number of attributes. To verify the applicability of the tool in comparison to standard computer and screen usage, they conducted a controlled experiment with program comprehension tasks comparing the tools *Eclipse*, *Code2City*, and *Code2CityVR* [RCE+19a]. In contrast to *Code2CityVR*, we base our visualization on dynamic instead of static analysis and present the behavior of an observed software system to the user. Baum et al. [BSK+17] present *Getaviz*, a tool for the empirical evaluation of software visualizations. The tool implements the concept of generative and model-driven software visualizations and allows to generate

## 11. Related Work

different representations of a software system. In detail it supports four major visualization metaphors including the city metaphor. When applying the city metaphor, an observed software system is represented by districts and included classes by buildings. Methods and Attributes as well as different metrics of this metaphor differ depending on the chosen option which can either be original, panels, bricks or floors. Furthermore, Getaviz also supports to display the visualizations in immersive VR. On the contrary we focus on structure and dynamic behavior of an observed software system, especially the communication between classes.

Evostreets [SKR19] is a visualization tool employing the city metaphor to support the comprehension of evolving software systems. The visualization displays streets orthogonally in relation to the hierarchy of the software system. Additionally, streets corresponding to higher levels within the hierarchy are wider than those corresponding to lower levels of it. Based on Evostreets, Steinbeck et al. presented a VR extension for Evostreets and performed a controlled experiment comparing three environments – a 2D orthographic projection with keyboard and mouse, a 2.5D projection with keyboard and mouse, and finally a VR projection with HMDs and related controllers [SKR19]. Another experiment comparing the navigation within the VR plugin versus a 3D desktop visualization on a traditional display, keyboard and mouse setup was conducted in [RGK18]. Furthermore, they analyzed how EvoStreets was used and observed by the participants in the experiment [SKR20]. In comparison to Evostreets, our approach focuses on the structure and dynamic behavior of an observed software system instead of its evolution and related dependencies. Additionally, we support teleportation as a option to navigation within the virtual environment. CodeHouse [HKI19] is a tool that visualizes the source code of a software in a virtual environment. The tool supports the comprehension of the structure of a software system on the one hand and offers to debug it and thus understand the behavior on the other hand. In contrast to our work, they arrange software components (i.e., modules) on the inside of a virtual cylinder to reduce line overlap between modules and. Additionally, our approach does not provide a capability to debug a software. Khaloo et al. [KMT+17] visualize software systems in VR by employing a park



metaphor. Their tool CodePark visualizes rooms which represent classes in a park based on their directory structure. In each room an overview of the methods of a class and their source code are displayed on a wallpaper. Furthermore, the tool offers a bird's eye view mode and a first-person view mode to explore the visualization. In comparison to CodePark our VR approach focuses on the dynamic behavior of a software system and provides a first-person view mode.

In addition to the presented VR approaches and tools there also exist related approaches employing other display and interaction devices. A selection is presented in the following. PerfVis [MHB+19] is a tool to visualize software performance through immersive augmented reality (AR). The approach employs an AR device (Microsoft HoloLens) to display the visualization and extends an IDE on a computer screen. For the city visualization they utilized the tool CodeCity [WL08] as introduced in Section 3.2. In contrast, we provide a VR based approach and focus on system and program comprehension of observed software systems, which is also capable to provide a small set of performance metrics. XRaSE [MSK+19b] is another tool which utilizes AR to provide comprehension, architecture analysis, and debugging capabilities for software systems. The tool offers a set of visualizations for several use cases and focuses, like our *ExplorViz* VR approach on the collaboration of developers. Compared to XRaSE, our VR approach concentrates on collaborative system and program comprehension but does not offer the option to debug a software system. SourceVis is a collaborative, multi-user software visualization tool usable on large multi-touch tables [AMN+13]. The tool offers multiple visualization techniques and supports the comprehension of software structures based on the visualization of software metrics and source code evolution. By applying multi-touch tables the tool is suitable for collaborative software visualization applications. Conversely to SourceVis, our collaborative, multi-user VR approach addresses the structure and dynamic behavior of a software system. Additionally, it is applicable by users, which are located in different locations.

## 11. Related Work

### 11.3 Database Visualization

There are several approaches, which are related toward our envisioned database visualization approach as presented in Section 7.4. Thus, we restrict our related work to approaches focusing on the visualization of databases. A selection of closely related approaches and tools is presented in the following.

*ExplorViz Legacy* [Flo16] is a web-based tool, which enables the monitoring and visualization of large software landscapes. Basically, it offers two different visualization options. The first visualization shows a reconstructed software landscape based on monitoring information. The second visualization offers an in-detail representation of a single application and involved packages, classes, and communication. Within the second visualization, executed database queries based on JDBC are listed. In contrast, our approach focuses on the monitoring and visualization of databases and applications, which communicate with them, to facilitate software landscape and database comprehension. DAHLIA [MC14] is an interactive visualization tool, which facilitates to analyze the database usage to support software and database schema evolution. The tool collects snapshots of database schemas from a software repository based on static analysis and utilizes a 3D visualization for exploring the monitored evolution. Recently they released Version 2.0, which includes the support for Object-Relational-Mapping frameworks [MC16]. Based on this feature, the tool allows to analyze the evolution of a database over its lifetime more precisely. In contrast to DAHLIA, our approach utilizes dynamic and static analysis to obtain a live visualization of the database and executed database queries from associated applications. Additionally, our approach addresses operators and developers. *NakeDB* [CHP+08] represents a dynamic visualization tool for huge databases. The tool generates database schema visualizations, more precisely visual graphs with color coded objects and shapes, based on database dumps. It features dynamic searching and filtering techniques, offers several visualization options, and provides interaction capabilities, like zooming and panning. In contrast to *NakeDB*, our approach enables a live visualization of databases, shows executed queries from applica-

## 11.4. Modularization and Reengineering

tions, and provides an overview of the monitored software landscape. InspectIT is an open source application performance management (APM) tool, which provides a monitoring sensor for databases.<sup>1</sup> Based on the gathered information, it is possible to perform a performance analysis on executed database queries. Although, analyzing the performance of executed database queries is an important use case within our approach, we draw our primary attention on the visualization and employed perspectives, to aid the comprehension process.

### 11.4 Modularization and Reengineering

In the area of software engineering, there are many papers that perform a software modernization in other contexts, e.g., software maintenance or reverse engineering. Thus, we restrict our related work to approaches, which focus on the modernization of monolithic applications towards a microservice architecture. Compared to frequently performed software modernizations, we did not reconstruct the underlying software architecture, since it was not our goal to keep the obsolete monolithic architecture provided by *GWT* within our application of *CORAL* to *ExplorViz Legacy*. Furthermore, we did not need to apply multiple refactoring iterations to modernize our software system. Instead, we successfully performed three iterations of our modularization and modernization process *CORAL* to continuously improve our software architecture and collaborative development process. For an comparison of existing software migration frameworks, we refer to [SIA+17].

There are several approaches to move legacy software systems to a software product line approach [LC13]. Our approach differs from these, since we do not just focus on the migration of a single legacy software system, but on the modularization and modernization of a software system and its related development process within *CORAL*. Villamizar et al. [VGC+15] evaluate monolithic and microservice architectures regarding the development and cloud deployment of enterprise applications. Their approach

---

<sup>1</sup><https://github.com/inspectIT/inspectIT>

## 11. Related Work

addresses similar elements to our modernization process. They employ modern technologies for separating microservices, e.g., Java in the backend and JS in the frontend, like we described Section 10.2. Contrary to their results, we did not face any of the mentioned problems during the migration, like failures or timeouts. In [ECA+16] Escobar et al. present an approach regarding the challenges of the modernization of legacy J2EE applications. They employ static and dynamic code analysis [HFG+11] to reconstruct architectural diagrams, which then can be used as a starting point during a modernization process. In contrast to our approach there was no need for us to reconstruct the software architecture, because we wanted to modernize it from the beginning due to previously mentioned drawbacks. Thus, we split our application based on our knowledge into several microservices and developed a communication concept based on a message broker. Carrasco et al. [CBD18] present a survey of architectural smells during the modernization towards a microservice architecture. They identified nine common pitfalls in terms of bad smells and provided potential solutions for them. *ExplorViz Legacy* was also covered by this survey and categorized by the “Single DevOps toolchain” pitfall. This pitfall concerns the usage of a single toolchain for all microservices. Fortunately, we addressed this pitfall since their observation during their survey by employing independent toolchains by means of pipelines within our continuous integration system for the backend and frontend microservices. Knoche and Hasselbring [KH18] describe a migration process to decompose an existing software system into several microservices. Additionally, they report from their gained experiences towards applying their presented approach in a legacy modernization project. Although their modernization drivers and goals are similar to our procedure, their approach features a more abstract point of view on the modernization process. Furthermore, they focus on programming language modernization and transaction systems. In [HS17] the authors present an industrial case study concerning the evolution of a long-living software system, namely a large e-commerce application. The addressed monolithic legacy software system was replaced by a microservice-based system. Compared to our approach, this system was completely rebuilt without retaining code from the (commercial) legacy software system. Our focus is to facilitate the collaborative development of open source software and also addresses

## 11.5. Collaboration in Software Engineering

the development process. We successfully developed our pipeline towards CI for all microservices (as mentioned in Section 10.2.5) to minimize the release cycles and offer development snapshots. Hasselbring et al. [HRJ+04] present an experience report for the migration of legacy business software systems towards a multi-tier architecture. Based on an architectural pattern they describe an approach to perform a partial duplication of business logic among a legacy system and newly deployed application server. Although the separation-of-concerns principle is violated, their approach offers a high degree of flexibility in the migration process. In contrast to their approach, we provide a holistic reengineering and modularization process focusing on the comprehension of such software systems. Furthermore, we also strive for a collaborative development process.

## 11.5 Collaboration in Software Engineering

The process of software engineering requires many software engineers to coordinate their efforts to design, implement, and maintain a large software system. Whitehead [Whi07] stated in 2007 that *"...at present there is no integrated web-based environment that covers the entire software development lifecycle, with existing tools typically covering a single phase, such as requirements, or UML diagramming."* His statement is still valid 13 years later if we take a look at available tools in software engineering. But there have emerged some promising approaches in the recent years, which improve the situation in a remarkable manner. For a comparison of recent collaborative development tools we refer to [LEP+10]. Meanwhile, a large set of concepts and tools exist in research and industry, which greatly address and support collaboration in software engineering in several aspects. In the following, we present a selection of relevant concepts and tools, which are complementary to our presented *CORAL* approach.

Software configuration management (SCM) systems depict the fundamental basis for coordinating file-based collaboration between software

## 11. Related Work

engineers. A few years ago SCM systems like Subversion<sup>2</sup> or Git<sup>3</sup> were often centrally hosted at the site of a software development company and only accessible with matching desktop applications. Nowadays, several cloud-based platforms like *Github*<sup>4</sup> or *Gitlab*<sup>5</sup> have emerged and offer a web-based, collaborative frontend on top of SCMs like Git. Furthermore, they do not provide only SCM capabilities but also additional collaboration features, e.g., project management, bug tracking, and release management. Thus, the number of employed tools for specific use cases can be reduced and the usage eased for developers. These tools are complementary to our *CORAL* approach, as they focus on other aspects of collaborative software engineering.

Another notable area of collaboration in software engineering are approaches that offer knowledge for developers like the question and answer platform Stack Overflow.<sup>6</sup> The platform provides questions and answers on a very broad range of topics in computer programming aspects and serves as a valuable knowledge base for specific issues. Thus, even difficult technical problems can be solved in a short time. While using our *CORAL* approach, developers can use such related platforms to increase their knowledge particular in technical aspects, which could be useful while employing and exploring technologies, frameworks, tools, and even software architectures.

CASE (computer-aided software engineering) tools provide visualization and automation capabilities to support software engineering in designing and generating source code at hand. Although a majority of them offer an integration of SCMs and handle concurrent access, very few support collaboration at hand. Especially designing software systems and related static and dynamic aspects in a collaboratively manner depicts an import aspect. For an overview of collaborative UML tools we refer to [BTE+14].

---

<sup>2</sup><https://subversion.apache.org>

<sup>3</sup><https://git-scm.com>

<sup>4</sup><https://github.com>

<sup>5</sup><https://about.gitlab.com>

<sup>6</sup><https://stackoverflow.com>

## 11.5. Collaboration in Software Engineering

Furthermore, there also exist several communication-oriented tools which facilitate the collaboration of developers. Examples are instant messaging services like Slack,<sup>7</sup> Mattermost,<sup>8</sup> or Discord,<sup>9</sup> which also provides a VoIP application platform. While using our *ExplorViz VR* approach, we employ Discord to facilitate the communication between our co-located users while they share their virtual environment.

Finally, visualization tools can support collaborative software engineering. Examples are the previously described XRaSE [MSK+19b], which utilizes AR to provide a collaborative comprehension, architecture analysis, and debugging capabilities for software systems, and SourceVis [AMN+13], which offers a collaborative, multi-user software visualization tool usable on large multi-touch tables to support the comprehension of software structures based on the visualization of software metrics and source code evolution.

---

<sup>7</sup><https://slack.com>

<sup>8</sup><https://mattermost.com>

<sup>9</sup><https://discord.com>





## **Part IV**

# **Conclusions and Future Work**



# Conclusions

In this thesis, we presented our collaborative reengineering and modularization approach *CORAL*. To verify the approach, we conducted an extensive case study on *ExplorViz* and performed multiple iterations, moving from a monolithic towards a microservice architecture. Furthermore, we developed an alternative display and interaction concept for the software city metaphor on the basis of VR named *ExplorViz VR*. Additionally, we performed multiple case studies with the VR approach as part of the *Tool-based Analysis* action within our *CORAL* approach. Last but not least, we presented our *RACCOON* approach for enabling live database visualization. In the following, we summarize the thesis, beginning with wrapping up our *CORAL* approach in Section 12.1. Afterwards, we conclude our presented collaborative, multi-user VR approach *ExplorViz VR* in Section 12.2. Then, we resume our live database visualization approach *RACCOON* in Section 12.3. Finally, we provide an overview of our evaluation results for our performed case studies in Section 12.4.

## 12. Conclusions

# Previous Publications

Parts of this chapter are already published in the following works:

1. Christian Zirkelbach. “Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems.” In: *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*. Hanover, Germany, 2017
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019
3. Christian Zirkelbach and Wilhelm Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Research Report 1901. Kiel University, Feb. 2019
4. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
5. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
6. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49

## 12.1 The CORAL Approach

We developed our tool-emplying *CORAL* approach with the purpose to support developers in the modularization and modernization of software systems in an iterative manner. The approach consists of five, consecutive actions – starting with the analysis of an observed software system, divided into a *Manual Analysis* and a *Tool-based Analysis*, and ending with the evaluation of an executed recommendation plan. An existing software project and involved systems, which are in need of modularization and modernization are analyzed within the *Manual Analysis* action first. Therefore, we need to take a look at the underlying architecture, employed technologies, and tools. This task includes a software architecture and modernization evaluation, to identify and reassess legacy source code, frameworks and utilized libraries, and execution environments. The software architecture evaluation task is divided into four parts – (i) a software architecture review, (ii) the application of a software architecture evaluation method, (iii) the identification of technical debt, and (iv) the examination of employed technologies and frameworks. Afterwards, the *Tool-based Analysis* action is performed, which aids the modularization process by detecting (technical) flaws, possible shortcomings, and optimization potential. In detail, we focus on the aspect of understanding the software system. We address this aspect by employing the software visualization tool *ExplorViz* with developed extensions to aid the system and program comprehension process. We employ *ExplorViz* to achieve a better understanding of a software system, which we want to modularize and modernize with our *CORAL* approach. By utilizing *ExplorViz* for the comprehension process, we take advantage of software visualizations instead of software artifacts like source code or documentation. Thus, we can enhance our previously obtained knowledge about the software system from discussions and interviews with the software developers. Within the *Recommendation* action, we study the observations from our analysis actions and design a recommendation plan in collaboration with the developers. The recommendation plan addresses possible (target) architectures, technologies, and frameworks. Furthermore, we aim for a collaborative development process. After evaluating the presented recommendation plan, the *Execution* action needs to take place. More precisely, we work out a

## 12. Conclusions

proof-of-concept implementation of the recommendation plan first. Thus, we can verify if the necessary technical adaptations are suitable to perform the reengineering and modularization process afterwards on a solid basis. Refactoring tools can help with a realization of the recommendation plan. Once we executed the recommendation plan, we need to perform an *Evaluation*, to verify its impact on the software system. Therefore, we focus on comparing the software quality based on metrics provided by software quality tools on the one hand and the software architecture through visual comparison on the other hand. Software quality tools like *Sonatype* and *SonarCloud* allow us to evaluate if the software quality has been improved after a performed iteration of the *CORAL* approach. Typically, the results of the evaluation are not sufficient after only one execution. Thus, it is likely, that the overall approach needs to be conducted multiple times to achieve an acceptable state.

Unlike previous work, we offer a holistic approach based on established concepts, best-practices, and state-of-the-art technologies. Furthermore, we employ several tools which support specific actions within our approach. Most important are software visualization tools like *ExplorViz* which help us to improve the system and program comprehension process on the one hand in the *Tool-based Analysis* action, and software quality tools within the *Evaluation* action.

### 12.2 The *ExplorViz* VR Approach

Additionally, we realized a collaborative, multi-user VR approach as an extension to *ExplorViz*. The VR approach is applied within the *Tool-based Analysis* action as part of our *CORAL* approach. The approach complements the already existing visualizations provided by *ExplorViz* and offers an alternative and immersive user experience for the system and program comprehension process. Since 2017, we continuously extended and improved the VR approach. Starting with a single-user VR approach as an alternative to the existing application-level visualization of *ExplorViz* on the screen, we realized a first multi-user VR approach in 2019, and achieved a fully

### 12.3. The *RACCOON* Approach

collaborative, multi-user VR approach offering a landscape-level perspective in 2020. The latest version allows multiple users with HMDs and related controllers to collaboratively explore and comprehend monitored software systems in VR. Based on several gestures mapped to related controllers the users are able to interact with the immersive software visualization and thus can perform system and program comprehension tasks in teams. As software development takes more and more place on a globally scale, we also considered this aspect within the software and allowed a location-independent usage. Thus, developers located at different sites are able to utilize our approach for several tasks.

In comparison to related work, we offer a truly collaborative and location-independent multi-user VR approach, which not only facilitates system and program comprehension in teams, but also encourages its application based on excellent usability and functionality. Furthermore, the web-based VR approach builds upon the WebVR interface specification, which allows a platform-independent usage with several recent HMDs and related input devices.

## 12.3 The *RACCOON* Approach

Furthermore, we presented another software visualization approach, named *RACCOON*, to support the comprehension of database behavior in software systems. Databases are essential components within large software landscapes, since they are employed in almost every information system. Based on the growing complexity of software systems and a steadily increasing amount of data which is collected, processed, and stored in databases, it is difficult to obtain a live overview of these software landscapes. *RACCOON* embodies a live visualization approach of databases and associated communication for large software landscapes. The employed visualization offers two different views – a landscape-level and a database-level perspective. The landscape-level perspective provides an overview of monitored applications and related databases. The database-level perspective reveals database schemas within a database, shows contained tables and relation-

## 12. Conclusions

ships, and allows for the inspection of executed database queries based on the monitoring information collected at runtime. Based on the visualizations, developers and operators are able to investigate the actual usage of databases by applications and thus support such users in performance and comprehension tasks. Since this is still a work in progress, we have not realized a proof-of-concept implementation and consecutive evaluation yet.

### 12.4 Evaluation Results

With our evaluation results, we showed that we successfully fulfilled the research goals of this thesis. More precisely, we presented two approaches in detail within our thesis – our *CORAL* approach on the one hand and our collaborative, multi-user VR approach *ExplorViz VR* on the other hand. The implementation of the latter approach is available as open source on *Github*.<sup>1</sup> We evaluated the approaches with a GQM-based approach with the previously described primary research questions in Chapter 5.

To answer the questions, we applied literature reviews, proof-of-concept implementations, and conducted several empirical experiments and case studies as research methods. In order to evaluate our *CORAL* approach itself, we conducted an extensive case study on the software visualization tool *ExplorViz*. More precisely, we performed multiple iterations of our approach while moving from a monolithic towards a microservice architecture. On the basis of *ExplorViz Legacy*, we employed our iterative, collaborative modularization and reengineering approach *CORAL* as a guidance through our modularization. More precisely, we performed three successful iterations to *ExplorViz Legacy* until we reached a sufficient state. After our first iteration, we realized our modularization process and architecture in terms of a proof-of-concept implementation and evaluated it afterwards by the development of several extensions of *ExplorViz*. Each of these extensions was developed by students and evaluated afterwards, in each case by at least a usability study. The results showed an overall high usability of each extension. In the case of our developed application discovery extension, we integrated it into our core project based on the high quality of the

---

<sup>1</sup><https://github.com/ExplorViz>



extension in addition to the good usability and time-saving aspect when instrumenting applications with *Kieker*. As the results of the modularization process were not sufficient yet, we performed a second iteration featuring a first microservice architecture. More precisely, the iteration led to several independent deployable services bundled with inter-service communication handled via the message broker *Kafka* and requests from the frontend towards the backend are passed through our reverse-proxy in form of *NGINX*. Furthermore, we enhanced our development and build process towards a more collaborative manner. Unfortunately, we were not satisfied with the results of the second iteration, because some services were still very large and poorly maintainable. Thus, we needed to perform a further decoupling of them. Additionally, we recognized that our release management and CI processes, as well as our documentation, still needed to be improved. Consequently, with these drawbacks in mind, we performed a third iteration, after which we achieved a fully decoupled microservice architecture, consisting of a set of self-contained systems and well-defined interfaces in-between. The inter-service communication is still handled through *Kafka*. Additionally, we replaced our reverse-proxy with *Traefik* for handling requests from the frontend towards the backend. For the release management and documentation, we further optimized our CI pipeline regarding *Docker* images and supplemental (API) documentation for developers and users. Since we successfully applied our *CORAL* approach towards *ExplorViz* with our case study, we showed that the approach is capable of supporting modernization and modularization processes of software systems. Thus, our goal G1 of the thesis (see Chapter 5) is fulfilled.

Within our thesis, we additionally developed our collaborative, multi-user VR approach *ExplorViz VR* in form of an extension for *ExplorViz* as a display and interaction alternative to the classical screen and keyboard and mouse setup. We designed a concept for the approach and realized it afterwards with an implementation. To evaluate the VR approach, we conducted three, consecutive empirical lab experiments with test subjects. First, we evaluated the initial version of our single-user VR approach focusing on usability aspects. In particular the navigation of our VR approach was accepted very well and felt intuitive for the subjects. The user experience and interaction

## 12. Conclusions

aspects of our approach also received good evaluations, but leaves room for improvement. In combination with our observations we imply that our VR approach combines the advantages of 3D visualizations with navigation and interaction capabilities in a virtual environment. In our second evaluation, we conducted an experiment to verify the user experience and usability of our collaborative, multi-user VR approach. Again, the interaction and user experience of our improved VR approach were accepted very well and felt intuitive. The visual appearance aspect of our approach also received a good evaluation, but leaves room for improvement. Although *Spectating* proved not to be a useful feature, we imply that our VR approach offers an intuitive and practical handling for the presented 3D software visualization. The experiment gave a first, successful impression of the collaborative capability of our multi-user VR approach. Finally, we performed a third experiment to investigate the applicability of our multi-user VR approach for program comprehension tasks, especially in teams. Based on the results of the experiment, we conclude that our multi-user VR extension is applicable for collaborative program comprehension tasks in the context of static and dynamic software analysis. We learned from our previous experiments and implemented a list of useful features, which enhanced the usability and applicability of our approach. Minor visualization issues regarding the visibility of the communication lines depending on the status of opened or closed components misled some subjects and thus resulted in incorrect answers. As a consequence, this aspect needs to be addressed in an upcoming version. All three empirical lab experiments showed that the VR approach successfully offers an alternative display and interaction concept for system and program comprehension using the software city metaphor. Furthermore, it offers an alternative, immersive visual approach, which provides a collaborative comprehension process applicable by distributed teams. Especially the high correctness rates within our third experiment, which focused on the applicability of the VR approach for program comprehension tasks in teams, provided impressive results. Thus, our goals G2 and G3 of the thesis (see Chapter 5) are fulfilled.

# Future Work

In this chapter, we present possible future work for our presented thesis and related approaches. First, we describe work regarding the holistic modularization and reengineering approach *CORAL*. Afterwards possible future work for our collaborative, multi-user VR approach *ExplorViz VR* is listed. Finally, further work regarding our presented database-behavior approach *RACCOON* is described.

### 13. Future Work

## Previous Publications

Parts of this chapter are already published in the following works:

1. Christian Zirkelbach. “Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems.” In: *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*. Hanover, Germany, 2017
2. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019
3. Christian Zirkelbach and Wilhelm Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Research Report 1901. Kiel University, Feb. 2019
4. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel University, Apr. 2019
5. Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019
6. Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. “The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software.” In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020), pages 34–49

## 13.1 The CORAL Approach

In the future, we are planning to further evaluate our successful application of CORAL to *ExplorViz*, especially in terms of developer collaboration. Additionally, we plan to move from our CI pipeline towards a Continuous Delivery (CD) environment. Thus, we expect to further decrease the interval between two releases and allow users to try out new versions, even development snapshots, as soon as possible. This aspect is also supported by our ADAMMS [KZH18] approach within *ExplorViz*, which eases the monitoring configuration of *Kieker* and thus the overall setup and deployment of *ExplorViz*. Furthermore, we plan to use architecture recovery tools like [GCF+17] for refactoring or documentation purposes in upcoming versions of *ExplorViz*. Recently, we applied *ExplorViz* within a case study, where we successfully performed a microservice decomposition with static and dynamic analysis of a monolithic application [KZH+20]. As a result, we plan to investigate, if we could enhance our CORAL approach with the applied decomposition process for future projects.

Maintainability and extensibility are essential aspects within our presented CORAL approach. Therefore, another possible future work depicts conducting a modularization case study applied to *ExplorViz*. Although we already evaluated the extensibility by providing a plugin-architecture and developing several extensions in [ZKH20], we strive for an evaluation focusing specific on modularization.

Employing benchmarking tools like MooBench [WEH15] could provide additional, valuable insights as part of our CORAL approach. The benchmarking could take place in addition to the *Software Quality Improvements* and *Software Architecture Comparison* steps within the *Evaluation* action. Thus, we would be able to verify if changes made had an impact on the performance of the related software system. Finally, further case studies applying the CORAL approach, particular in industrial collaborations, should be conducted.

## 13. Future Work

### 13.2 The *ExplorViz* VR Approach

The results of the latest experiment applying our collaborative, multi-user VR approach *ExplorViz* VR showed its applicability for software comprehension tasks in the context of static and dynamic software analysis, especially in teams. Minor visualization issues regarding the visibility of the communication lines depending on the status of opened or closed components misled some subjects and thus resulted in incorrect answers. After a first analysis, we were able to trace these problems back to navigational and layout issues. As a consequence, this aspect needs to be addressed in an upcoming version. Furthermore, three specific features could enhance the usability and applicability of the approach. First, fixing the freezing problem during the rendering, which could appear in large software landscapes, would greatly improve the VR user experience. Second, the visualization of traces and the option to follow them could also increase the applicability. Last but not least, an interactive tutorial would facilitate an easier access to the virtual environment and interaction capabilities.

One crucial future work embodies the migration of the VR extension to the WebXR<sup>1</sup> specification, because the currently employed WebVR<sup>2</sup> specification became deprecated in recent months. Our VR approach should further be evaluated with successor HMDs, which could offer higher display resolutions and thus enable an improved perceived immersion, or provide alternative interaction capabilities.

Additionally, further experiments should be conducted to investigate alternative scenarios in the context of software comprehension. More precisely, we are interested, if the collaborative feature of our approach offers an advantage over working alone. Thus, we could conduct a controlled experiment where we compare the work of teams with single users on comprehension tasks. Finally, future work should replicate our experiments with professional software engineers.

---

<sup>1</sup><https://immersiveweb.dev>

<sup>2</sup><https://webvr.info>

## 13.3 The RACCOON Approach

Future work lies in the realization of a proof-of-concept implementation in combination with a consecutive evaluation. Although, there are still some open questions left, which need to be addressed upfront:

- ▷ Which layout is suitable for our landscape-level perspective to represent the applications and databases within a large software landscape?
- ▷ How do we link databases and related artifacts (e.g. deployed software) for our landscape-level perspective?
- ▷ How do we relate database queries with business code – where in the source code was the query started? (package-/class-level)
- ▷ Which advantage does our database-level perspective offer over traditional database diagrams like the entity relationship diagrams?
- ▷ To what extent is our approach also applicable for distributed and NoSQL databases [BCD12]?
- ▷ How can we successfully combine our database monitoring and visualization approach with existing Application Performance Management tools?
- ▷ Which related approaches or tools could be employed, when evaluating our approach within a controlled experiment?

Finally, since we already offer a VR visualization of the software city metaphor in our *ExplorViz VR* approach, we could also provide a 3D database visualization in this context. According to the evaluation results of our VR approach *ExplorViz VR*, an application of VR could be useful to provide a more immersive database comprehension.





**Part V**

# **Appendix**



## *ExplorViz* VR Evaluation - Questionnaires

This appendix contains the employed questionnaires for our *ExplorViz* VR evaluations in this thesis. In Appendix A.1 we depict the questionnaire for our single-user VR usability study (see Section 9.2). Afterwards, we present the questionnaire for our distributed multi-user VR usability study (see Section 9.3.1) in Appendix A.2. Finally, in Appendix A.3, we provide the questionnaire for our distributed multi-user VR applicability experiment (see Section 9.3.2).

## A. *ExplorViz* VR Evaluation - Questionnaires

### A.1 Single-User VR Usability Study - Questionnaire

Sehr geehrter Proband/sehr geehrte Probandin,

wir bedanken uns im Vorfeld der Studie für Ihre Teilnahme und möchten Ihnen ein paar allgemeine Informationen geben.

Diese Studie ist anonym und es werden keine persönlichen Informationen nach außen geliefert. Uns ist weiterhin wichtig, dass Sie zur Kenntnis nehmen, dass wir Sie bitten unser Produkt zu bewerten. Aus diesem Grund haben wir uns im Vorfeld gegen Laufzeit-Aufnahmen von Ihnen (z.B. Bild und Ton) entschieden. Wir zeichnen lediglich den Bildschirm auf, der keine Verbindung zu Ihrer Person ermöglicht. Wir werden Sie bitten bestimmte Aufgaben zu lösen und uns am Ende einen Fragebogen zu beantworten.

Im Laufe dieser Studie werden Sie mit zwei Technologien in Berührung kommen: Der VR-Brille und dem Brain-Computer-Interface.

Bei der VR-Brille handelt es sich um eine Brille, die dem Nutzer einen 3-dimensionalen Raum aufzeigt, in dem der Nutzer interagieren, sich wie gewohnt umschauen und bewegen kann. Der Fußboden dieses Raums entspricht dem realen Raum, der für Bewegungen zur Verfügung steht und sollte deshalb nicht verlassen werden. In dieser Studie wird Ihnen in diesem Raum eine Darstellungsform einer Software (oder auch Computerprogramm) gezeigt. Wir werden diese als Landschaft bezeichnen. Es ist kein Vorwissen bzgl. eines Programms gefordert. Wir werden lediglich geometrische Fragen stellen. Die dargestellte Landschaft besteht aus Systemen, die als graue Boxen dargestellt sind. Diese Boxen enthalten Teilsysteme, die durch Öffnen dieser Boxen dargestellt werden. Eine geöffnete Box wird als Fläche dargestellt, auf der sich die Teilsysteme befinden. Die Teilsysteme werden als grüne Boxen oder blaue Flächen dargestellt. Aus den blauen Flächen können besondere Teilsysteme erzeugt werden. Diese werden als Applikationen bezeichnet und bestehen ebenfalls aus Boxen und Flächen. Die Interaktion mit den Boxen und Flächen erfolgt durch zwei Controller. Diese verfügen im virtuellen Raum über einen Strahl, der wie ein Laserpointer funktioniert. Mit dem Strahl anvisierte Boxen und Flächen werden rot markiert, wenn mit ihnen interagiert werden kann. Zu jedem anvisierten Objekt der Landschaft oder Applikation kann ein Informationstext angezeigt werden.

Das Brain-Computer-Interface (BCI) ist ein neuro-technisches Stirnband, welches die Spannung auf der Kopfoberfläche misst und darauf reagiert. Dadurch ist das Gerät begrenzt in der Lage bestimmte Gedanken wieder zu erkennen und darauf zu reagieren. Nach einer kurzen Trainingsphase sollen Sie das BCI nutzen, um ein paar simple Aufgaben zu lösen.

# A.1. Single-User VR Usability Study - Questionnaire

## Allgemeine Angaben zur Person

---

ID: \_\_\_\_

Studiengang: \_\_\_\_\_, Fachsemester \_\_\_\_\_

Lerntyp: \_\_\_\_\_

*Lernen durch Schreiben, Lesen etc.*

Abschluss : \_\_\_\_\_

*Bei bereits abgeschlossenem Studium*

	stimme völlig zu	stimme eher zu	neutral	stimme eher nicht zu	stimme gar nicht zu
Ich fühle mich fit und gesund	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich bin anfällig für Motion Sickness (Reisekrankheit, Seekrankheit, etc.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich habe eine Sehschwäche	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich trage eine Brille	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich lerne schnell	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich werde leicht nervös	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich bin aufgeregt	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich kann mich gut konzentrieren	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich stehe unter dem Einfluss von Koffein	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich stehe unter dem Einfluss von Nikotin	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich stehe unter dem Einfluss von Medikamenten	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich habe bereits Erfahrung mit virtueller Realität gemacht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich habe bereits Erfahrung mit Brain-Computer-Interfaces gemacht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## A. ExplorViz VR Evaluation - Questionnaires

### Notizen zum Ablauf (VR)

---

ID: \_\_\_\_\_

	stimme völlig zu	stimme eher zu	neutral	stimme eher nicht zu	stimme gar nicht zu
Die VR-Brille war einfach anzubringen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Der Proband hat das höchste System gefunden	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Allgemeine Bemerkungen:

---

---

---

---

Verlauf:

---

---

---

---

Besonderheiten:

---

---

---

---

## A.1. Single-User VR Usability Study - Questionnaire

### Virtuelle Realität (VR)

	stimme völlig zu	stimme eher zu	neutral	stimme eher nicht zu	stimme gar nicht zu
Das Erkunden der virtuellen Umgebung war angenehm	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Darstellung der virtuellen Umgebung war angenehm	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die VR-Brille war komfortabel zu tragen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Das Umschauen im virtuellen Raum durch Drehen des Kopfes oder Drehen des Körpers war intuitiv	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Annäherung an Objekte durch Daraufzugehen oder durch Annähern des Kopfes war intuitiv	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Gewinnung von Abstand zu Objekten durch Wegbewegen war intuitiv	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Fortbewegung im virtuellen Raum durch reales Gehen war praktisch	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Körper- und Kopfbewegungen zur Navigation (umschauen, annähern, distanzieren) waren leicht zu lernen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Interaktion mit der virtuellen Umgebung durch die Controller war intuitiv	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Bedienung der Controller war leicht zu lernen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Das automatische Markieren von Objekten durch den Strahl des Controllers war praktisch	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Durch die Markierung habe ich schnell verstanden mit welchen Objekten interagiert werden kann	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Das Öffnen und Schließen von Boxen durch den Abzug war praktisch	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Das Rotieren, Verschieben und Zoomen eines Objektes durch Gedrückthalten des großen runden Knopfes war praktisch	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Beim Rotieren, Verschieben oder Zoomen eines Objektes hatte ich das Gefühl, dieses in der Hand zu halten	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Darstellung von Informationen in einem Fenster neben dem Controller war praktisch	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich konnte dieses Fenster ähnlich wie eine Zeitung in die Hand nehmen und meinem Blickwinkel anpassen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Rückmeldung der virtuellen Umgebung war direkt und flüssig	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich war frei von Schwindel und Übelkeit	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich hätte noch weitere Zeit in der virtuellen Umgebung verbringen können	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich würde den VR-Modus wiederverwenden	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## A. *ExplorViz* VR Evaluation - Questionnaires

Anmerkungen:

---

---

---

Verbesserungsvorschläge:

---

---

---



## A.2 Distributed Multi-User VR Usability Study - Questionnaire

Dear participant,

we thank you very much for your participation in this experiment. You and another participant are about to test and evaluate a collaborative virtual reality (VR) extension to ExplorViz. You are free to quit the evaluation whenever you are not feeling comfortable anymore. First of all, we would like you to answer some general questions about you on the next page. This will allow us to add some context to the results later on. Your answers and test results are anonymous. Then we are proceeding by introducing you to ExplorViz and the input devices for VR. When you have learned the basics about ExplorViz and the controls you will be asked to solve some exercises together with your partner. Lastly, you will be asked to rate certain aspects about the user experience. This will help us improve the extension and tell us how well ExplorViz can be used with multiple users in virtual reality.

## A. ExplorViz VR Evaluation - Questionnaires

### 1 General Personal Data

ID: ..... *Vive*  *Rift*

Age: .....

Profession: *student*  *researcher*  *other*

Subject of study: ..... *bachelor*  *master*

Gender: *male*  *female*  *diverse*

Do you wear glasses? *yes*  *no*

Do you have any visual impairment? *yes*  *no*

If so, which: .....

	0	+	++	++ +
Experience with objectoriented programming	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Experience with ExplorViz	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Experience with VR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Are you claustrophobic?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Are you afraid of heights?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Do you suffer from seasickness?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
How well do you know the other proband?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## A.2. Distributed Multi-User VR Usability Study - Questionnaire

### 2 Introduction

ExplorViz is a monitoring and visualization software for large software landscapes. ExplorViz uses two different views for the visualization which are shown simultaneously with the VR extension. The landscape view is a view of a software landscape and is particularly suitable to get an overview of landscapes. Here you can see systems (grey), servers (green) and the software running on the servers (blue). The communication between software is represented by orange lines, where the thickness of the lines correlates with the number of calls it represents.

The application view represents a three-dimensional model of the software. On top of a grey foundation software packages (components) are shown in green, which in turn can contain components or individual classes (blue). The height of the blue blocks indicates the number of objects belonging to the class. Here, too, the communication between objects is visualized with orange lines. You can select individual classes or call up additional information for a class.

This type of representation is intended to be a metaphor for a three-dimensional city, with the classes here representing (high-)buildings and communication between classes are streets.

Now that you are familiar with the concepts we would like you to get familiar with the controls. On the next page there is an overview of the functionalities of all buttons on the controllers you are about to use. Please use the HTC Vive or Oculus Rift respectively now while we guide you through all control options. Feel free to ask questions throughout the experiment of something is unclear to you.

## A. *ExplorViz* VR Evaluation - Questionnaires

### 3 Tasks

1. Connect via the menu.
2. Change your height as you like.
3. Open the user list.
4. Move and rotate the landscape as you like.
5. Find the node '10.0.2.2'.
6. Find the application 'Wiki' and try to open it.
7. With how many applications does 'Webshop' communicate?
8. Open the application 'Webshop'.
9. Move and rotate the application 'Webshop' as you like.
10. Mark the class 'ItemHelper' which is part of the application 'Webshop'.
11. How many active instances has the class 'ImplementationHandler'  
(located in component org/webshop/kernel/impl)?
12. Open a second application.
13. Use the spectate feature (only as long as you are comfortable using it).
14. The spectated proaband may mark the component labeling in Webshop.
15. Quit spectating and spectate each other with reversed roles.
16. Mark component connector in DatabaseConnector.
17. Quit spectating.
18. Close all open system and components.
19. Disconnect via the menu.

## A.2. Distributed Multi-User VR Usability Study - Questionnaire

### 4 Rating

	--	-	+	++
Moving and rotating the landscape is well realized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The moving and rotating of the applications is well realized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The movement (incl. teleportation) in the virtual space is intuitive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The structure of the menu is intuitive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
Navigation through the menu is intuitive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The visualization of other users is well done	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The movements of the other user were displayed to me without delay	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
I had the impression that I was in the same room with the other user (positions and state of users, landscape & applications were synchronized)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The number of text insertions was reasonable	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				

## A. ExplorViz VR Evaluation - Questionnaires

	--	-	+	++
Text overlays were clearly readable .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Text overlays were visually appealing (duration, length, position, color, size, animation) .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Highlighting objects is a useful feature .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ExplorViz with VR extension is suitable for team work .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
During the experiment (except spectator mode) I felt nausea or something alike .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
While spectating I felt nausea or something of the like .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I would use the spectator mode again .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I would use ExplorViz with the VR extension again .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## A.2. Distributed Multi-User VR Usability Study - Questionnaire

Proposals for future features

Improvement proposals

Further remarks

## A. *ExplorViz* VR Evaluation - Questionnaires

### A.3 Distributed Multi-User VR Applicability Experiment - Questionnaire

#### ExplorViz - Collaborative VR Experiment

##### 1. Introduction

First of all we would like to thank you for participating in the study.

In this experiment we want to evaluate the applicability and usefulness of the Collaborative Multi-User Virtual Reality (VR) extension of ExplorViz for system and program comprehension. ExplorViz is a tool for live monitoring and visualization of large software landscapes. Subject of the study is the Common Component Model Example (CoCoME) – a community case study for component-based software engineering and software evolution.

We will start with a short introduction to ExplorViz with a practical VR-based tutorial upfront. Afterwards, you will work together with a second person as a team to solve tasks with a varying degree of difficulty.

You will apply ExplorViz in order to answer questions to assess their comprehension on the CoCoME application. Based on this study, we plan to draw conclusions on the applicability and usefulness of the VR approach for system and program comprehension.

If you have further questions, please don't hesitate to ask. You can cancel your participation in the study at any time!

Before we start with the questionnaire, we want to illustrate the CoCoME software system you are going to examine. The system resembles a software system for a supermarket chain. Please watch the attached short video.





## A.3. Distributed Multi-User VR Applicability Experiment - Questionnaire

### 2. General Information

Before we proceed to the questions of this study, please give us a few information about your person. Those will only be used to analyze the results of the experiment.

#### 1. Subject Id (as listed on the registration paper) \*

#### 2. Team Id (as listed on the registration paper) \*

#### 3. Team: Relationship \*

	Not at all	Barely	Well	Very Well
I know my team member...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

#### 4. Team: Origin \*

	None specific	Work	Personal
I know my team member from...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

#### 5. Team: Experience \*

	Never	Sometime	Often
I have worked with my team member before...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

#### 6. Gender \*

#### 7. Age \*

## A. ExplorViz VR Evaluation - Questionnaires

8. Kind of Occupation \*

- Student/Grad. Student
- Industry
- Research

9. Highest completed degree \*

Choose...

10. Affiliation (University, Company, ...) \*

11. Expertise \*

Choose...

12. Prior Experience \*

	None	Beginner	Intermediate	Advanced	Expert
Java, C++, or similar	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Architecture	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Development in Teams	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Static Software Analysis	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Dynamic Software Analysis	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Program Comprehension	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reengineering & Reverse-Engineering	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Virtual Reality (VR)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
ExplorViz	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
CoCoME	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

13. Do you have any color vision impairments? \*

Choose...

## A.3. Distributed Multi-User VR Applicability Experiment - Questionnaire

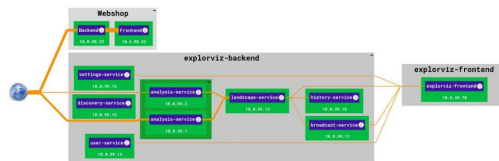
### 3. ExplorViz Introduction

#### ExplorViz Introduction

ExplorViz is a monitoring program that serves to graphically visualize large software landscapes. The tool provides two perspectives, the landscape perspective and the application level perspective.

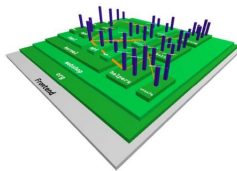
#### Landscape-level Perspective

The more general landscape perspective shows systems in the form of grey boxes, which contain green colored nodes, e.g., servers. Nodes in turn contain applications (blue). The communication between those is represented by orange lines, which vary in thickness depending on the number of requests. There is a more detailed view for each application, the application level perspective.



#### Application-level Perspective

In the application level perspective, the packages and the elements they contain are displayed in a nested manner. Packages are displayed as green boxes. The lowest elements of this hierarchy, i.e., classes, are displayed as purple bodies, where the height of the object indicates the number of instances. Communication between classes is displayed in the same way as communication between applications.



## A. *ExplorViz* VR Evaluation - Questionnaires

Additional information is provided for individual entities of the visualization, such as CPU utilization for nodes or the average response time for communication lines. You can interact with the landscape in different ways. It is possible to open or close individual entities. In addition, entities can be color-coded to highlight them.

### A.3. Distributed Multi-User VR Applicability Experiment - Questionnaire

#### 4. Tutorial

Thank you for completing the first part of the questionnaire.  
As we want to continue with the VR tutorial, please say that your finished to your examiner.

## A. *ExplorViz* VR Evaluation - Questionnaires

### 5. Tutorial

#### **Before the tutorial**

1. Load tutorial demo
2. Connect both users
3. Open key assignment
4. Start screen recording via Xbox Gamebar

#### **Tutorial**

1. You are both in the same virtual room. You can move within it by moving in physical space.
2. There is a limited space in which tracking is possible. But you will recognize the room boundaries when you are about to leave the room.
3. You can look around in virtual space by moving your head.
4. You can interact with the environment by using the controllers. The controllers are also displayed in virtual space. On the controllers are laser pointers that you use to point to objects that you want to interact with.
5. From the other user you see the headset and the controllers in the virtual room.
6. On the left controller you should now see the key assignment.
7. You are able to move in greater distances by teleporting.
8. Teleport yourself so that you have a good view of the grey box on the ground.
9. The grey box on the floor is the system you are supposed to analyse. Open the system.
10. In the system there are several servers (here green boxes), on which applications are shown in blue.
11. The symbol indicates that this is Java code.
12. Open one application each.
13. Move the application so that you can see the components clearly. Look at the key mapping for help and hold down the corresponding key.

### A.3. Distributed Multi-User VR Applicability Experiment - Questionnaire

14. The green boxes on the grey area are packages, they contain hierarchically arranged further packages and classes.
  15. When you move the laser pointers over the objects, they are colored differently for the duration to show that you can perform an operation on them.
  16. A darker object with the right laser pointer means that you can open or close the object.
  17. A darker object with the left laser pointer means that you can select the object.
  18. Now select the package "net".
  19. The select function is used to highlight an object in color. Only one object per user can be selected at a time.
  20. By selecting another object or by clicking again, the selected object is deselected.
  21. You can also see the selection of the other user.
  22. Each user has his own color to select, the color corresponds to the color in which the username or the laser pointer can be seen. Look at the other user to verify the color.
  23. If you want to display more information about the package "net", look at the key mapping for help.
  24. With this overview you can get additional data. For example, it shows the number of packages and classes.
  25. The selection and the retrieval of this info view happen independently.
  26. You can close this overview by opening the overview for another object instead or by clicking in the empty room.
  27. Now open the "net" package and the packages it contains step by step until you can see all 6 classes.
  28. Classes are shown as blue bars, the height of the class corresponds to the number of instances.
  29. Now select the class "SQLStatementHandler".
  30. Call additional information about "SQLStatementHandler", you will find out the exact number of instances and the called methods here.
  31. The orange lines between the classes represent the communication, whereby the width of the lines depends on the number of requests.
  32. Select the largest communication line you see directly in front of you.
32. Select the largest communication line you see directly in front of you.
  33. The communication always refers to classes. Open all necessary packages to see the second affected class.
  34. Now call up additional information, here you see above between which classes the communication takes place.
  35. The arrow in between shows which class calls which. In case of bidirectional communication there are 2 arrows.
  36. Below you can see the actual number of requests and the average response time.
  37. By clicking the red ball with the white X next to the application, you can close the application view.

## A. ExplorViz VR Evaluation - Questionnaires

### 6. Software Structure

#### Task 1: Structure of the Software System

##### Instructions

The first task refers to the **structure** of software systems. As they are often composed of a huge number of packages, classes, interfaces, and methods, it can be difficult to understand the structure and the depending functionality. In order to understand the structure, you have to (i) get to know the involved packages and (ii) how the software system is assembled.

This task requires you to understand the structure and hierarchy of the observed software system CoCoMe.

16. Name all top-level packages within the application CoCoME. Top-level packages are packages on the highest hierarchy level. \*

17. Name all classes in the package 'printer'. The package 'printer' can be found under the package 'org.cocome.cloud'. \*

18. Compare the packages 'context' and 'userdisplay'. Which of them contains more classes? The package 'context' can be found under the package 'org.cocome.tradingsystem'. \*

### 7. System Load Inspection

#### Task 2: System Load Inspection

##### Instructions

The second task is about identifying bottlenecks in the system at runtime. Analyze the communication between packages and classes in terms of number of requests.

19. 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. \*

20. 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. \*

21. 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. \*



## A.3. Distributed Multi-User VR Applicability Experiment - Questionnaire

8. Evolution

### Task 3: Evolution of the Software System

#### Instructions

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.

22. 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'. \*

## A. ExplorViz VR Evaluation - Questionnaires

### 9. Conclusion

#### 23. Please rate the perceived difficulty of the tasks. \*

	very easy	easy	difficult	very difficult
Task 1: Structure of the Software System	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 2: System Load Inspection	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 3: Evolution of the Software System	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

#### 24. Please rate how much the following questions apply to you. \*

	strongly disagree	disagree	agree	strongly agree
I felt like I was in the same room with my team member.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt I could communicate and interact well with my team member.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt working on Task 1 as a team was helpful.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt working on Task 2 as a team was helpful.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt working on Task 3 as a team was helpful.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I consider the VR mode of ExplorViz to be suitable for analyzing software.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I consider the VR mode suitable for working in a team.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

#### 25. Which features would you consider helpful for collaborative use within the VR extension in ExplorViz? Which ones could be improved or are missing?

#### 26. Please give us some additional feedback, suggestions, and comments.

# List of Figures

2.1	Reengineering and related processes in the context of software engineering [CC90]. . . . .	24
2.2	An overview of the technical debt landscape [KNO12]. . . . .	26
2.3	The Architecture Business Cycle [BCK03]. . . . .	27
3.1	A street from Software World [KM99]. . . . .	32
3.2	CodeCity: visualizing a structural overview of the software Findbugs [WLR10]. . . . .	33
3.3	<i>ExplorViz Legacy</i> : Landscape-level perspective [Flo16; FRH15].	34
3.4	<i>ExplorViz Legacy</i> : Application-level perspective [Flo16]. . . . .	35
3.5	Human and virtual environment interaction loop [BM07]. . . . .	37
4.1	Blind monks examining an elephant, an ukiyo-e print by Hanabusa Itchō (1652–1724). . . . .	40
4.2	Overview of model-driven collaboration [WMG+10]. . . . .	42
6.1	UML activity diagram illustrating the <i>CORAL</i> method . . . . .	57
7.1	Landscape-level perspective of <i>ExplorViz</i> , Version 1.3.0 . . . . .	71
7.2	Application-level perspective of <i>ExplorViz</i> , Version 1.3.0 . . . . .	72
7.3	UML activity diagram showing the program comprehension workflow using <i>ExplorViz</i> . . . . .	73
7.4	Visualized 3D application-level perspective: Simplified version of the Java application <i>Neo4j</i> in VR [ZKH19a]. . . . .	80
7.5	VR movement gestures (Leap Motion Controller). . . . .	84
7.6	VR interaction gestures (Leap Motion Controller). . . . .	85
7.7	VR movement gestures (left controller – HTC Vive). . . . .	86
7.8	VR interaction gestures (right controller – HTC Vive). . . . .	87

## List of Figures

7.9	Architecture of <i>ExplorViz</i> in Version 1.5.0 with our realized VR extension showing used software services with employed frameworks and libraries. . . . .	88
7.10	VR gestures (Leap Motion Controller). . . . .	90
7.11	VR gestures (HTC Vive). . . . .	91
7.12	Sketch showing two users employing the Multi-User VR approach collaboratively based on [Häs17]. . . . .	94
7.13	VR interaction gestures (left controller – HTC Vive (Pro)). . . . .	99
7.14	VR interaction gestures (left controller – Oculus Touch). . . . .	100
7.15	VR interaction gestures (right controller – HTC Vive (Pro)). . . . .	101
7.16	VR interaction gestures (right controller – Oculus Touch) . . . . .	102
7.17	Overview of our <i>RACCOON</i> approach for live database behavior visualization. . . . .	104
7.18	Landscape-level visualization: communication between systems, applications, and databases in a software landscape. . . . .	107
7.19	Mockup of the database-level visualization: tables, relationships, and executed queries in a single database ( <i>Core (DB2)</i> ). . . . .	110
7.20	Semantics of an opened table: columns and constraints. . . . .	111
7.21	Visualized Opened table <i>Product</i> – showing columns, constraints, and relationships. . . . .	111
9.1	Landscape-level perspective showing the <i>PubFlow</i> demo landscape of <i>ExplorViz</i> in Version 1.2.0. . . . .	128
9.2	Application-level perspective showing the simplified Neo4J application within the demo landscape of <i>ExplorViz</i> in Version 1.2.0. . . . .	129
9.3	Usability study results for <i>General Questions</i> ( $D_1$ - $D_4$ ). . . . .	131
9.4	Usability study results for <i>User Experience</i> questions ( $A_1$ - $A_7$ ). . . . .	132
9.5	Usability study results for <i>Navigation</i> questions ( $B_1$ - $B_5$ ). . . . .	133
9.6	Usability study results for <i>Interaction</i> questions ( $C_1$ - $C_9$ ). . . . .	134
9.7	Usability study leader observation results ( $S_1$ - $S_2$ ). . . . .	136
9.8	Average agreement rates for the question categories A - C. . . . .	141
9.9	Landscape-level perspective showing the <i>SampleApplication</i> landscape visualized in <i>ExplorViz</i> in Version 1.3.0. . . . .	146

## List of Figures

9.10	Application-level perspective showing the minimalist application <i>SampleApplication</i> within the same named landscape visualized in <i>ExplorViz</i> in Version 1.3.0. . . . .	147
9.11	Usability study results for <i>General Questions</i> ( $G_1$ - $G_7$ ). . . . .	148
9.12	Usability study results for <i>Interaction</i> questions ( $A_1$ - $A_5$ ). . . . .	150
9.13	Usability study results for <i>User Experience</i> questions ( $B_1$ - $B_4$ ). . . . .	151
9.14	Usability study results for <i>Visual Appearance</i> questions ( $C_1$ - $C_6$ ). . . . .	152
9.15	Usability study results for <i>Spectating</i> questions ( $D_1$ and $D_2$ ). . . . .	153
9.16	Average agreement rates for the question categories <i>A - D</i> . . . . .	159
9.17	Landscape-level perspective showing the <i>CoCoME</i> landscape visualized in <i>ExplorViz</i> in Version 1.5.0. . . . .	163
9.18	Application-level perspective showing the <i>CoCoME</i> application within the same-named landscape visualized in <i>ExplorViz</i> in Version 1.5.0. . . . .	164
9.19	Landscape-level perspective showing two instance of the <i>SampleApplication</i> within the tutorial landscape visualized in <i>ExplorViz</i> in Version 1.5.0. . . . .	166
9.20	Application-level perspective showing the first instance of the <i>SampleApplication</i> application within the tutorial landscape visualized in <i>ExplorViz</i> in Version 1.5.0. . . . .	167
9.21	Experiment results for <i>Experience Questions</i> ( $E_1$ - $E_{10}$ ). . . . .	168
9.22	Experiment results for <i>Assigned Tasks</i> ( $A_{1,1}$ - $A_{3,1}$ ). . . . .	170
9.23	Experiment results for the <i>Perceived Task Difficulty</i> of our task categories ( $A_1 - A_3$ ). . . . .	171
9.24	Experiment results for the <i>Feedback Questions</i> . . . . .	172
10.1	Architectural overview and employed software stack of the monolithic <i>ExplorViz Legacy</i> . . . . .	187
10.2	Architectural overview and software stack of the modularized <i>ExplorViz</i> (after the first iteration). . . . .	191
10.3	Screenshot of the application discovery extension of <i>ExplorViz</i> showing three detected applications on a monitored server. . . . .	195
10.4	Screenshot of the architecture conformance checking (ACC) extension of <i>ExplorViz</i> showing two checked systems. . . . .	196

## List of Figures

10.5 Architectural overview and software stack of <i>ExplorViz</i> (after the second iteration). . . . .	199
10.6 Current architectural overview and software stack of <i>ExplorViz</i> (after the third iteration). . . . .	202

# List of Tables

8.1	Evaluation: conducted case studies and their related goals applying <i>CORAL</i> . . . . .	117
8.2	Evaluation: conducted case studies and their related goals for developed extensions. . . . .	119
8.3	Evaluation: conducted case studies and their related goals for database comprehension and employing <i>ExplorViz</i> . . . . .	121
9.1	Hardware setup employed for the multi-user VR approach evaluation. . . . .	127
9.2	Hardware setup employed for the usability multi-user VR approach evaluation. . . . .	144
9.3	Hardware setup employed for the applicability multi-user VR approach evaluation. . . . .	162





# Listings

7.1	Excerpt of an example aop.xml configuration file of Kieker. . .	76
7.2	Excerpt of an example kieker.monitoring.properties configuration file of Kieker. . . . .	77



# Bibliography

- [AAE16] N. Alshuqayran, N. Ali, and R. Evans. "A Systematic Mapping Study in Microservice Architecture." In: *Proceedings of the 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. Nov. 2016 (cited on page 191).
- [AAV+19a] V. L. Averbukh, N. V. Averbukh, P. Vasev, I. I. Gvozdarev, G. I. Levchuk, I. Starodubtsev, and M. Forghani. "Virtual reality as an instrument of computer visualization." In: *Proceedings of the International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*. 2019 (cited on page 5).
- [AAV+19b] Vladimir Averbukh, Natalya Averbukh, Pavel Vasev, Ilya Gvozdarev, Georgy Levchuk, Leonid Melkozerov, and Igor Mikhaylov. "Metaphors for software visualization systems based on virtual reality." In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Edited by Lucio Tommaso De Paolis and Patrick Bourdot. 2019 (cited on page 5).
- [AC15] Fahem Ahmed and Luiz Fernando Capretz. "A framework for process assessment of software product line." In: *Journal of Information Technology Theory and Application* 7.1 (2015) (cited on page 62).
- [AD16] R Vijay Anand and M Dinakaran. "Popular agile methods in software development: review and analysis." In: *International Journal of Applied Engineering Research* 11.5 (2016) (cited on page 64).
- [AGW+16] C. Anthes, R. J. Garcia-Hernandez, M. Wiedemann, and D. Kranzlmüller. "State of the art of virtual reality technology." In: *Proceedings of the IEEE Aerospace Conference*. Mar. 2016 (cited on page 36).

## Bibliography

- [AMN+13] C. Anslow, S. Marshall, J. Noble, and R. Biddle. "Sourcevis: collaborative software visualization for co-located environments." In: *Proceedings of the First IEEE Working Conference on Software Visualization (VISSOFT)*. Sept. 2013 (cited on pages 215, 221).
- [Apa20] Apache Software Foundation. *Apache Kafka*. Aug. 31, 2020. URL: <https://kafka.apache.org> (cited on page 198).
- [ASB+19] M. Ali Babar, H. Shen, S. Biffl, and D. Winkler. "An empirical study of the effectiveness of software architecture evaluation meetings." In: *IEEE Access* 7 (2019). DOI: 10.1109/ACCESS.2019.2922265 (cited on page 66).
- [Awa05] MA Awad. "A comparison between agile and traditional software development methodologies." In: (2005) (cited on page 64).
- [Bal99] Thomas Ball. "The concept of dynamic analysis." In: *Software Engineering — ESEC/FSE '99*. Edited by Oscar Nierstrasz and Michel Lemoine. 1999 (cited on page 23).
- [BCD12] P. Bagade, A. Chandra, and A. B. Dhende. "Designing performance monitoring tool for nosql cassandra distributed database." In: *Proceedings of the International Conference on Education and e-Learning Innovations*. July 2012. DOI: 10.1109/ICEELI.2012.6360579 (cited on page 237).
- [BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. 2003 (cited on page 27).
- [BD13] Fabian Beck and Stephan Diehl. "Visual comparison of software architectures." In: *Information Visualization* 12.2 (2013). DOI: 10.1177/1473871612455983. URL: <https://doi.org/10.1177/1473871612455983> (cited on page 65).
- [BG09] M. A. Babar and I. Gorton. "Software architecture review: the state of practice." In: *Computer* 42.7 (July 2009). DOI: 10.1109/MC.2009.233 (cited on pages 28, 59).

- [BGH06] Stefan Biffl, Paul Grünbacher, and Michael Halling. "A family of experiments to investigate the effects of groupware for software inspection." In: *Automated Software Engineering* 13.3 (July 2006). DOI: 10.1007/s10851-006-8531-5. URL: <https://doi.org/10.1007/s10851-006-8531-5> (cited on page 66).
- [BH13] Peer Christoph Brauer and Wilhelm Hasselbring. "Pubflow: a scientific data publication framework for marine science." In: *Proceedings of the International Conference on Marine Data and Information Systems (IMDIS 2013)*. Volume 54. Sept. 2013. URL: <http://eprints.uni-kiel.de/22399/> (cited on page 127).
- [BHJ+18] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. "Microservices migration patterns." In: *Software: Practice and Experience* 48.11 (2018). DOI: 10.1002/spe.2608. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2608> (cited on page 63).
- [BHJ16] A. Balalaie, A. Heydarnoori, and P. Jamshidi. "Microservices architecture enables devops: migration to a cloud-native architecture." In: *IEEE Software* 33.3 (2016) (cited on page 63).
- [BK01] S. Bassil and R. K. Keller. "Software visualization tools: survey and analysis." In: *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*. May 2001. DOI: 10.1109/WPC.2001.921708 (cited on page 4).
- [BM02] Erin Bradner and Gloria Mark. "Why distance matters: effects on cooperation, persuasion and deception." In: *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*. 2002. DOI: 10.1145/587078.587110. URL: <https://doi.org/10.1145/587078.587110> (cited on page 44).
- [BM07] D. A. Bowman and R. P. McMahan. "Virtual Reality: How Much Immersion Is Enough?" In: *Computer* 40.7 (2007) (cited on pages 36, 37).

## Bibliography

- [BN12] L. Bass and R. L. Nord. "Understanding the context of architecture evaluation methods." In: *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture 2012*. Aug. 2012. DOI: 10.1109/WICSA-ECSA.2012.45 (cited on pages 28, 59).
- [Bro99] F. P. Brooks. "What's real about virtual reality?" In: *IEEE Computer Graphics and Applications* 19.6 (1999) (cited on page 36).
- [Brü20] Johannes Brück. "Collaborative program comprehension based on virtual reality." Bachelor thesis. Kiel University, Apr. 2020. URL: <http://eprints.uni-kiel.de/49581/> (cited on pages 17, 119, 160, 165).
- [BSK+17] David Baum, Jan Schilbach, Pascal Kovacs, Ulrich Eisenecker, and Richard Müller. "Getaviz: generating structural, behavioral, and evolutionary views of software systems for empirical evaluation." In: *Proceedings of the IEEE Working Conference on Software Visualization (VISOFT)*. IEEE. Sept. 2017 (cited on page 213).
- [BSP13] O. Benomar, H. Sahraoui, and P. Poulin. "Visualizing software dynamicities with heat maps." In: *Proceedings of the First IEEE Working Conference on Software Visualization (VISOFT)*. Sept. 2013. DOI: 10.1109/VISSOFT.2013.6650524 (cited on page 212).
- [BT06] Rafael Ferreira Barcelos and Guilherme Horta Travassos. "Evaluation approaches for software architectural documents: a systematic review." In: *CIbSE*. 2006 (cited on pages 28, 59).
- [BTE+14] Carlos Eduardo Barbosa, Glauco Trindade, Vanessa Janni Epelbaum, Juliana Gomes Chang, Jonice Oliveira, José A. Rodrigues Neto, and Jano Moreira de Souza. "Challenges on designing a distributed collaborative UML editor." In: *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 2014. DOI: 10.1109/CSCWD.2014.6846817 (cited on page 220).

- [BW84] V. R. Basili and D. M. Weiss. "A Methodology for Collecting Valid Software Engineering Data." In: *IEEE Transactions on Software Engineering* SE-10.6 (Nov. 1984). DOI: 10.1109/TSE.1984.5010301 (cited on pages 49, 50, 125, 183).
- [BWS+06] Nelis Boucké, Danny Weyns, Kurt Schelfhout, and Tom Holvoet. "Applying the atam to an architecture for decentralized control of a transportation system." In: *Quality of Software Architectures*. Edited by Christine Hofmeister, Ivica Crnkovic, and Ralf Reussner. 2006 (cited on pages 28, 60).
- [CAH+19] P. Cruz, H. Astudillo, R. Hilliard, and M. Collado. "Assessing migration of a 20-year-old system to a micro-service platform using atam." In: *Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2019. DOI: 10.1109/ICSA-C.2019.00039 (cited on pages 28, 60).
- [Car99] Mackinlay Card. *Readings in information visualization: using vision to think*. 1999 (cited on page 30).
- [CBD18] Andrés Carrasco, Brent van Bladel, and Serge Demeyer. "Migrating Towards Microservices: Migration and Architecture Smells." In: *Proceedings of the 2nd International Workshop on Refactoring*. 2018 (cited on pages 191, 218).
- [CC90] E. J. Chikofsky and J. H. Cross. "Reverse engineering and design recovery: a taxonomy." In: *IEEE Software* 7.1 (1990) (cited on pages 24, 25).
- [CD07] G. Canfora and M. Di Penta. "New frontiers of reverse engineering." In: *Future of Software Engineering (FOSE '07)*. 2007 (cited on page 25).
- [CDC11] Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. "Achievements and Challenges in Software Reverse Engineering." In: *Communications of the ACM* 54.4 (Apr. 2011). DOI: 10.1145/1924421.1924451 (cited on page 23).

## Bibliography

- [Che76] Peter Pin-Shan Chen. “The Entity-Relationship Model – Toward a Unified View of Data.” In: *ACM Trans. Database Syst.* 1.1 (Mar. 1976). DOI: 10.1145/320434.320440 (cited on pages 6, 103, 108).
- [CHP+08] Luis Miguel Cortes-Pena, Yi Han, Neil Pradhan, and Romain Rigaux. *NakeDB: Database Schema Visualization*. 2008. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.460.5562> (cited on page 216).
- [CKK+03] Paul Clements, Rick Kazman, Mark Klein, et al. *Evaluating software architectures*. 2003 (cited on pages 28, 59).
- [CMT16] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. “Workload characterization: a survey revisited.” In: *ACM Comput. Surv.* 48.3 (Feb. 2016). DOI: 10.1145/2856127. URL: <https://doi.org/10.1145/2856127> (cited on page 77).
- [CMZ+20] Noptanit Chotisarn, Leonel Merino, Xu Zheng, Supaporn Lonpalawong, Tianye Zhang, Mingliang Xu, and Wei Chen. “A systematic literature review of modern software visualization.” In: *Journal of Visualization* 23.4 (June 2020). DOI: 10.1007/s12650-020-00647-w. URL: <http://dx.doi.org/10.1007/s12650-020-00647-w> (cited on page 209).
- [CO11] Paul Clarke and Rory V. O’Connor. “An approach to evaluating software process adaptation.” In: *Software Process Improvement and Capability Determination*. Edited by Rory V. O’Connor, Terry Rout, Fergal McCaffery, and Alec Dorling. 2011 (cited on pages 58, 62).
- [CPD+16] Bruno Costa, Paulo F. Pires, Flávia C. Delicato, and Paulo Merson. “Evaluating rest architectures—approach, tooling and guidelines.” In: *Journal of Systems and Software* 112 (2016). DOI: <https://doi.org/10.1016/j.jss.2015.09.039>. URL: <http://www.sciencedirect.com/science/article/pii/S0164121215002150> (cited on pages 28, 60).



- [CSJ+16] T. H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora. "Finding and Evaluating the Performance Impact of Redundant Data Access for Applications that are Developed Using Object-Relational Mapping Frameworks." In: *IEEE Transactions on Software Engineering* 42.12 (Dec. 2016). DOI: 10.1109/TSE.2016.2553039 (cited on page 109).
- [Cun92] Ward Cunningham. "The wycash portfolio management system." In: *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*. 1992. DOI: 10.1145/157709.157715. URL: <http://doi.acm.org/10.1145/157709.157715> (cited on page 25).
- [CX16a] Chunyang Chen and Zhenchang Xing. "Mining technology landscape from stack overflow." In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2016. DOI: 10.1145/2961111.2962588. URL: <https://doi.org/10.1145/2961111.2962588> (cited on page 61).
- [CX16b] Chunyang Chen and Zhenchang Xing. "Similartech: automatically recommend analogical libraries across different programming languages." In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 2016. DOI: 10.1145/2970276.2970290. URL: <https://doi.org/10.1145/2970276.2970290> (cited on page 61).
- [CXH16] C. Chen, Z. Xing, and L. Han. "Techland: assisting technology landscape inquiries with insights from stack overflow." In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Oct. 2016. DOI: 10.1109/ICSME.2016.17 (cited on page 61).
- [CYC14] C. A. Cois, J. Yankel, and A. Connell. "Modern DevOps: Optimizing software development through effective system interactions." In: *Proceedings of the IEEE International Professional Communication Conference (IPCC)*. Oct. 2014. DOI: 10.1109/IPCC.2014.7020388 (cited on page 64).

## Bibliography

- [CZv+09] B. Cornelissen, A. Zaidman, A. van Deursen, and B. van Rompaey. "Trace visualization for program comprehension: a controlled experiment." In: *Proceedings of the IEEE 17th International Conference on Program Comprehension*. May 2009. DOI: 10.1109/ICPC.2009.5090033 (cited on page 4).
- [DGL+17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. "Microservices: Yesterday, Today, and Tomorrow." In: *Present and Ulterior Software Engineering*. 2017 (cited on page 191).
- [Die07] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. 1st edition. 2007 (cited on page 31).
- [DMM+15] T. Dal Sasso, R. Minelli, A. Mocci, and M. Lanza. "Blended, not stirred: multi-concern visualization of large software systems." In: *Proceedings of the IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. Sept. 2015. DOI: 10.1109/VISSOFT.2015.7332420 (cited on page 212).
- [DN10] Markus Dick and Stefan Naumann. "Enhancing software engineering processes towards sustainable software product design." In: *Integration of Environmental Information in Europe*. Edited by Klaus Greve and Armin B. Cremers. 2010 (cited on pages 58, 62).
- [DN90] Lionel E Deimel and J Fernando Naveda. *Reading Computer Programs: Instructor's Guide and Exercises*. Technical report. Software Engineering Institute, Carnegie Mellon University, 1990 (cited on page 29).
- [EBO+15] Neil A. Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L. Nord, and Ian Gorton. "Measure it? manage it? ignore it? software practitioners and technical debt." In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015. DOI: 10.1145/2786805.2786848 (cited on page 60).

- [ECA+16] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas. "Towards the understanding and evolution of monolithic applications as microservices." In: *Proceedings of the XLII Latin American Computing Conference (CLEI)*. Oct. 2016 (cited on page 218).
- [Ecl20] Eclipse Foundation. *AspectJ*. Version 1.8.5. Aug. 31, 2020. URL: <https://www.eclipse.org/aspectj> (cited on page 188).
- [Eic17] Felix Eichhorst. "Analyse der Microservices eines digitalen Marktplatzes mittels ExplorViz." Master thesis. Kiel University, Oct. 2017. URL: <http://eprints.uni-kiel.de/39982/> (cited on pages 14, 72, 78, 121).
- [EKP16] C. Ebert, M. Kuhrmann, and R. Prikladnicki. "Global software engineering: evolution and trends." In: *Proceedings of the IEEE 11th International Conference on Global Software Engineering (ICGSE)*. 2016 (cited on pages 44, 93).
- [Emb20] Ember Core Team. *Ember.js*. Version 3.6.0. Aug. 31, 2020. URL: <https://www.emberjs.com> (cited on page 192).
- [EPP15] A. Elliott, B. Peiris, and C. Parnin. "Virtual reality in software engineering: affordances, applications, and challenges." In: *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering*. Volume 2. May 2015. DOI: 10.1109/ICSE.2015.191 (cited on pages 5, 79).
- [ERS+14] B. Ens, D. Rea, R. Shpaner, H. Hemmati, J. E. Young, and P. Irani. "Chronotwigger: a visual analytics tool for understanding source and test co-evolution." In: *Proceedings of the Second IEEE Working Conference on Software Visualization*. Sept. 2014. DOI: 10.1109/VISSOFT.2014.28 (cited on page 211).
- [ESL20] ESLint Team. *ESLint*. Version 5.12.0. Aug. 31, 2020. URL: <https://eslint.org> (cited on page 200).
- [Exp20] ExplorViz Team. *ExplorViz Developer and User Documentation Wiki*. Aug. 31, 2020. URL: <https://github.com/ExplorViz/Docs/wiki> (cited on page 204).

## Bibliography

- [FBB+13] S. Fraser, J. Bishop, B. Boehm, P. Kathail, P. Kruchten, I. Ozkaya, and A. Szyrkarski. "Technical debt: past, present, and future (panel)." In: *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. 2013 (cited on page 25).
- [FFH+15] F. Fittkau, S. Finke, W. Hasselbring, and J. Waller. "Comparing trace visualizations for program comprehension through controlled experiments." In: *Proceedings of the IEEE 23rd International Conference on Program Comprehension*. May 2015. DOI: 10.1109/ICPC.2015.37 (cited on pages 70, 185).
- [FH11] Sören Frey and Wilhelm Hasselbring. "The cloudmig approach: model-based migration of software systems to cloud-optimized applications." In: *International Journal on Advances in Software* 4.3 and 4 (2011). URL: <http://oceanrep.geomar.de/14431/> (cited on page 63).
- [FH15] Florian Fittkau and Wilhelm Hasselbring. "Elastic application-level monitoring for large software landscapes in the cloud." In: *Service Oriented and Cloud Computing*. Edited by Schahram Dustdar, Frank Leymann, and Massimo Villari. 2015 (cited on page 77).
- [FHL02] Stefan Ferber, Peter Heidl, and Peter Lutz. "Reviewing product line architectures: experience report of atam in an automotive context." In: *Software Product-Family Engineering*. Edited by Frank van der Linden. 2002 (cited on pages 28, 60).
- [FKH15a] F. Fittkau, A. Krause, and W. Hasselbring. "Exploring software cities in virtual reality." In: *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*. 2015. DOI: 10.1109/VISSOFT.2015.7332423. URL: <http://dx.doi.org/10.1109/VISSOFT.2015.7332423> (cited on pages 5, 36, 79, 83, 89).
- [FKH15b] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. "Hierarchical software landscape visualization for system comprehension: a controlled experiment." In: *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*. Sept. 2015 (cited on pages 36, 70, 79, 185).

- [FLM18] P. Di Francesco, P. Lago, and I. Malavolta. "Migrating Towards Microservice Architectures: An Industrial Survey." In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*. Apr. 2018. DOI: 10.1109/ICSA.2018.00012 (cited on pages 4, 41).
- [Flo16] Florian Fittkau and Alexander Krause and Wilhelm Hasselbring. "Software Landscape and Application Visualization for System Comprehension with ExplorViz." In: *Information and Software Technology* (2016). DOI: {<http://dx.doi.org/10.1016/j.infsof.2016.07.004>} (cited on pages 6, 34, 35, 70, 103, 184, 216).
- [FRH15] Florian Fittkau, Sascha Roth, and Wilhelm Hasselbring. "Explorviz: visual runtime behavior analysis of enterprise application landscapes." In: *Proceedings of the 23rd European Conference on Information Systems (ECIS 2015)*. AIS. 2015 (cited on pages 34, 36, 70, 79, 184).
- [FRM+18] M. Franzago, D. D. Ruscio, I. Malavolta, and H. Muccini. "Collaborative model-driven software engineering: a classification framework and a research map." In: *IEEE Transactions on Software Engineering* 44.12 (2018) (cited on page 41).
- [FWW+13] Florian Fittkau, Jan Waller, Christian Wulf, and Wilhelm Hasselbring. "Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach." In: *Proceedings of the IEEE International Working Conference on Software Visualization (VISOFT 2013)*. Sept. 2013. URL: <http://eprints.uni-kiel.de/21840/> (cited on page 34).
- [Gar13] L. Garber. "Gestural technology: moving interfaces in a new direction." In: *Computer* 46.10 (2013). DOI: 10.1109/MC.2013.352. URL: <http://dx.doi.org/10.1109/MC.2013.352> (cited on pages 4, 79).
- [GCF+17] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle. "MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems." In: *Proceedings of the IEEE International Conference on Software*

## Bibliography

- Architecture Workshops (ICSAW)*. Apr. 2017 (cited on pages 206, 235).
- [GKB+18] N. Ghrairi, S. Kpodjedo, A. Barrak, F. Petrillo, and F. Khomh. "The state of practice on virtual reality (vr) applications: an exploratory study on github and stack overflow." In: *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS)*. July 2018. DOI: 10.1109/QRS.2018.00048 (cited on page 5).
- [GM17] K. Gupta and M. Mathuria. "Improving performance of web application approaches using connection pooling." In: *Proceedings of the International Conference of Electronics, Communication and Aerospace Technology (ICECA)*. Volume 2. Apr. 2017. DOI: 10.1109/ICECA.2017.8212833 (cited on page 77).
- [Gob14] C. Goble. "Better Software, Better Research." In: *IEEE Internet Computing* 18.5 (Sept. 2014). DOI: 10.1109/MIC.2014.88 (cited on page 3).
- [GSI15] Ishaya GAMBO, Abimbola SORIYAN, and Rhoda IKONO. "Health Information Systems' Architectural Evaluation with Architecture Trade-off Analysis Method (ATAM): a case study in Nigeria." In: *Journal of Health Informatics in Developing Countries* 9.1 (2015) (cited on pages 28, 60).
- [Hac18] Tim Hackel. "Architekturkonformitätsüberprüfung von Softwarelandschaften mittels ExplorViz." Diploma thesis. Kiel University, Sept. 2018. URL: <http://eprints.uni-kiel.de/44272/> (cited on pages 15, 66, 119, 196).
- [Han18] Malte Hansen. "Collaborative Software Exploration with the HTC Vive in ExplorViz." Bachelor thesis. Kiel University, Sept. 2018. URL: <http://eprints.uni-kiel.de/44435/> (cited on pages 15, 119, 142, 145).

- [Häs17] Timm Häsemeyer. "Kollaboratives Erkunden von Software mithilfe virtueller Realität in ExplorViz." Bachelor thesis. Kiel University, Sept. 2017. URL: <http://eprints.uni-kiel.de/39670/> (cited on pages 14, 94, 119, 126, 129).
- [Has18] Wilhelm Hasselbring. "Software architecture: past, present, future." In: *The Essence of Software Engineering*. Edited by Volker Gruhn and Rüdiger Striemer. 2018. DOI: 10.1007/978-3-319-73897-0\_10. URL: [https://doi.org/10.1007/978-3-319-73897-0\\_10](https://doi.org/10.1007/978-3-319-73897-0_10) (cited on pages 26, 59).
- [HB16] S. Hassan and R. Bahsoon. "Microservices and their design trade-offs: a self-adaptive roadmap." In: *Proceedings of the IEEE International Conference on Services Computing (SCC)*. June 2016. DOI: 10.1109/SCC.2016.113 (cited on page 4).
- [HCH+20] W. Hasselbring, L. Carr, S. Hettrick, H. Packer, and T. Tiropanis. "Open Source Research Software." In: *Computer* 53.8 (2020) (cited on page 3).
- [Her07] J. D. Herbsleb. "Global Software Engineering: The Future of Socio-technical Coordination." In: *Proceedings of Future of Software Engineering (FOSE '07)*. May 2007. DOI: 10.1109/FOSE.2007.11 (cited on pages 3, 44).
- [HFG+11] André van Hoorn et al. "Dynamod project: Dynamic analysis for model-driven software modernization." In: *Proc. of the 1st Int. Workshop on Model-Driven Software Migration (MDSM) 2011, Project Presentations*. Volume 708. 2011 (cited on page 218).
- [HH20] Wilhelm Hasselbring and André [van Hoorn]. "Kieker: a monitoring framework for software engineering research." In: *Software Impacts* 5 (2020). DOI: <https://doi.org/10.1016/j.simpa.2020.100019>. URL: <http://www.sciencedirect.com/science/article/pii/S2665963820300063> (cited on pages 74, 192).

## Bibliography

- [HHJ+13] Wilhelm Hasselbring, Robert Heinrich, Reiner Jung, Andreas Metzger, Klaus Pohl, Ralf Reussner, and Eric Schmieders. *lobserve: integrated observation and modeling techniques to support adaptation and evolution of software systems*. Research Report 1309. Christian-Albrechts-Universität Kiel, Oct. 2013. URL: <http://oceanrep.geomar.de/22077/> (cited on page 162).
- [HJZ+17] Robert Heinrich, Reiner Jung, Christian Zirkelbach, Wilhelm Hasselbring, and Ralf Reussner. “An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications”. In: *Software Architecture for Big Data and the Cloud*. Edited by Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim. June 2017. URL: <http://eprints.uni-kiel.de/36798/> (cited on pages 9, 62, 68, 70, 72, 77, 78, 121, 162, 185).
- [HKG+11] André van Hoorn, Holger Knoche, Wolfgang Goerigk, and Wilhelm Hasselbring. “Model-driven instrumentation for dynamic analysis of legacy software systems.” In: *Proceedings of the 13th Workshop Software-Reengineering (WSR 2011)*. May 2011. URL: <http://eprints.uni-kiel.de/14424/> (cited on page 192).
- [HKI19] A. Hori, M. Kawakami, and M. Ichii. “Codehouse: vr code visualization tool.” In: *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*. Sept. 2019. DOI: 10.1109/VISSOFT.2019.00018 (cited on page 214).
- [HRH+09] André van Hoorn, Matthias Rohr, Wilhelm Hasselbring, Jan Waller, Jens Ehlers, Sören Frey, and Dennis Kieselhorst. *Continuous monitoring of software services: Design and application of the Kieker framework*. Technical report TR-0921. Department of Computer Science, Kiel University, Germany, Nov. 2009 (cited on page 74).
- [HRJ+04] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, and S. Krieghoff. “The dublo architecture pattern for smooth migration of business information systems: an experience report.” In: *Proceedings of the 26th International Conference on Software Engineering*. 2004 (cited on page 219).



- [HRR+13] Israel Herraiz, Daniel Rodriguez, Gregorio Robles, and Jesus M. Gonzalez-Barahona. "The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review." In: *ACM Computing Surveys (CSUR)* 46.2 (Dec. 2013). DOI: 10.1145/2543581.2543595 (cited on page 3).
- [HRR16] Robert Heinrich, Kiana Rostami, and Ralf Reussner. *The Co-CoME platform for collaborative empirical research on information system evolution*. Technical report 2. Karlsruhe Institute of Technology, 2016 (cited on page 162).
- [HS17] W. Hasselbring and G. Steinacker. "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce." In: *Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW)*. Apr. 2017. DOI: 10.1109/ICSAW.2017.11 (cited on pages 3, 191, 218).
- [HTH+16] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig. "Usage, costs, and benefits of continuous integration in open-source projects." In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Sept. 2016 (cited on page 200).
- [HWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. "Kieker: a framework for application performance monitoring and dynamic software analysis." In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. 2012. DOI: 10.1145/2188286.2188326. URL: <https://doi.org/10.1145/2188286.2188326> (cited on pages 74, 192).
- [HZJ17] Robert Heinrich, Christian Zirkelbach, and Reiner Jung. "Architectural Runtime Modeling and Visualization for Quality-Aware DevOps in Cloud Applications." In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA 2017)*. 2017. URL: <http://eprints.uni-kiel.de/37439/> (cited on pages 9, 62, 68, 70, 72, 77, 78, 121, 162, 185).

## Bibliography

- [JA18] Reiner Jung and Marc Adolf. "The JPetStore Suite: A concise Experiment Setup for Research." In: *Symposium on Software Performance* (Nov. 2018). URL: <http://oceanrep.geomar.de/46706/> (cited on page 77).
- [JAP13] P. Jamshidi, A. Ahmad, and C. Pahl. "Cloud migration research: a systematic review." In: *IEEE Transactions on Cloud Computing* 1.2 (July 2013). DOI: 10.1109/TCC.2013.10 (cited on page 63).
- [JDP19] Sanjay Joshi, Bharat Deshpande, and Sasikumar Punnekkat. "Experimental analysis of dependency factors of software product reliability using sonarqube." In: *Joint Proceedings of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM Mensura 2019)*. Nov. 7, 2019 (cited on page 65).
- [JG12] Natalia Juristo and Omar S. Gómez. "Replication of software engineering experiments." In: *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*. Edited by Bertrand Meyer and Martin Nordio. 2012. DOI: 10.1007/978-3-642-25231-0\_2 (cited on pages 49, 126, 139, 142, 160).
- [JK07] Bernhard Jenny and Nathaniel Kelso. "Color Design for the Color Vision Impaired." In: *Cartographic Perspectives* 0.58 (2007) (cited on page 106).
- [Joy20] Joyent. *Node.js*. Version 10.15.0. Aug. 31, 2020. URL: <https://nodejs.org> (cited on page 192).
- [JPC+15] Pooyan Jamshidi, Claus Pahl, Samuel Chinenyeze, and Xiaodong Liu. "Cloud migration patterns: a multi-cloud service architecture perspective." In: *Service-Oriented Computing - IC-SOC 2014 Workshops*. Edited by Farouk Toumani, Barbara Pernici, Daniela Grigori, Djamel Benslimane, Jan Mendling, Nejib Ben Hadj-Alouane, Brian Blake, Olivier Perrin, Iman Saleh Moustafa, and Sami Bhiri. 2015 (cited on page 63).

- [Kan17] Maria-Anna Kandsorra. "Eye Tracking Based Experiments in ExplorViz." Master thesis. Kiel University, May 2017. URL: <http://eprints.uni-kiel.de/38102/> (cited on pages 13, 119).
- [KBE+12] Taimur Khan, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. "Visualization and evolution of software architectures." In: *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering-Proceedings of IRTG 1131 Workshop 2011*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012 (cited on page 31).
- [KDJ04] Barbara A. Kitchenham, Tore Dyba, and Magne Jorgensen. "Evidence-based software engineering." In: *Proceedings of the 26th International Conference on Software Engineering*. 2004. URL: <http://dl.acm.org/citation.cfm?id=998675.999432> (cited on pages 49, 126, 142, 160).
- [Ker05] Joshua Kerievsky. *Refactoring to patterns*. 2005 (cited on page 25).
- [KH18] H. Knoche and W. Hasselbring. "Using Microservices for Legacy Software Modernization." In: *IEEE Software* 35.3 (May 2018). DOI: 10.1109/MS.2018.2141035 (cited on pages 191, 218).
- [KH19] Holger Knoche and Wilhelm Hasselbring. "Drivers and barriers for microservice adoption – a survey among professionals in Germany." In: *Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modelling* 14.1 (2019). DOI: 10.18417/emisa.14.1 (cited on pages 4, 191).
- [KK16] Arif Ali Khan and Jacky Keung. "Systematic review of success factors and barriers for software process improvement in global software development." English. In: *IET Software* 10 (5 Oct. 2016). URL: <https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2015.0038> (cited on pages 44, 64).
- [KKC00] Rick Kazman, Mark Klein, and Paul Clements. *ATAM: Method for architecture evaluation*. Technical report. Carnegie-Mellon Software Engineering Institute, 2000 (cited on pages 28, 59).

## Bibliography

- [KKN+19] Arif Ali Khan, Jacky Keung, Mahmood Niazi, Shahid Hussain, and Mohammad Shameem. "Gsepim: a roadmap for software process assessment and improvement in the domain of global software development." In: *Journal of Software: Evolution and Process* 31.1 (2019). e1988 JSME-18-0098.R1. DOI: 10.1002/smr.1988. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1988> (cited on pages 62, 64).
- [KKY+13] K. Kobayashi, M. Kamimura, K. Yano, K. Kato, and A. Matsuo. "SArF map: Visualizing software architecture from feature and layer viewpoints." In: *Proceedings of the 21st International Conference on Program Comprehension (ICPC)*. May 2013. DOI: 10.1109/ICPC.2013.6613832 (cited on page 211).
- [KLN08] A. Kuhn, P. Loretan, and O. Nierstrasz. "Consistent Layout for Thematic Software Maps." In: *Proceedings of the WCRE*. 2008. DOI: 10.1109/WCRE.2008.45 (cited on page 106).
- [KM99] C. Knight and M. Munro. "Comprehension with[in] virtual environment visualisations." In: *Proceedings of the Seventh International Workshop on Program Comprehension*. 1999 (cited on pages 30–32).
- [KMT+17] P. Khaloo, M. Maghoubi, E. Taranta, D. Bettner, and J. Laviola. "Code park: a new 3d code visualization tool." In: *Proceedings of the IEEE Working Conference on Software Visualization (VIS-SOFT)*. Sept. 2017. DOI: 10.1109/VISSOFT.2017.10 (cited on page 214).
- [KN14] J. Knodel and M. Naab. "Software architecture evaluation in practice: retrospective on more than 50 architecture evaluations in industry." In: *Proceedings of the IEEE/IFIP Conference on Software Architecture*. Apr. 2014. DOI: 10.1109/WICSA.2014.37 (cited on pages 28, 58–60).
- [KN16] Jens Knodel and Matthias Naab. *Pragmatic evaluation of software architectures*. 2016 (cited on page 58).
- [KNO12] P. Kruchten, R. L. Nord, and I. Ozkaya. "Technical debt: from metaphor to theory and practice." In: *IEEE Software* 29.6 (Nov. 2012). DOI: 10.1109/MS.2012.167 (cited on pages 25, 26, 60).

- [Kön18] Daniel König. “Collaborative Software Exploration with the Oculus Rift in ExplorViz.” Bachelor thesis. Kiel University, Sept. 2018. URL: <http://eprints.uni-kiel.de/44434/> (cited on pages 15, 119, 142, 145).
- [Koz11] Heiko Koziolok. “Sustainability evaluation of software architectures: a systematic review.” In: *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*. 2011. DOI: 10.1145/2000259.2000263 (cited on pages 28, 58, 59).
- [KPP+02] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg. “Preliminary guidelines for empirical research in software engineering.” In: *IEEE Transactions on Software Engineering* 28.8 (Aug. 2002). DOI: 10.1109/TSE.2002.1027796 (cited on pages 49, 126, 142, 160).
- [Kra18] Alexander Krause. “Enterprise Application Discovery and Monitoring Management with ExplorViz.” Master thesis. Kiel University, Mar. 2018. URL: <http://eprints.uni-kiel.de/42625/> (cited on pages 14, 119).
- [Kri19] Florian Krippner. “Design und Implementierung eines Dashboards für ExplorViz.” Bachelor thesis. Kiel University, Sept. 2019. URL: <http://eprints.uni-kiel.de/47847/> (cited on pages 16, 119).
- [KSS+12] Lars Kristian Klauske, Christoph Daniel Schulze, Miro Spöemann, and Reinhard von Hanxleden. “Improved Layout for Data Flow Diagrams with Port Constraints.” In: *Proceedings of the 7th International Conference Diagrams, Canterbury, UK, July 2-6, 2012*. Edited by Philip Cox, Beryl Plimmer, and Peter Rodgers. 2012. DOI: 10.1007/978-3-642-31223-6\_11 (cited on page 106).
- [KWF+19] Jacob Krüger, Jens Wiemann, Wolfram Fenske, Gunter Saake, and Thomas Leich. “Program comprehension and developers’ memory.” In: *INFORMATIK 2019: 50 Jahre Gesellschaft für Informatik – Informatik für Gesellschaft*. Edited by Klaus

## Bibliography

- David, Kurt Geihs, Martin Lange, and Gerd Stumme. 2019. DOI: 10.18420/inf2019\_13 (cited on pages 30, 78).
- [KZH+20] Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, and Dan Kröger. “Microservice Decomposition via Static and Dynamic Analysis of the Monolith.” In: *Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2020. URL: <https://doi.org/10.1109/ICSA-C50368.2020.00011> (cited on pages 12, 70, 72, 121, 124, 182, 235).
- [KZH18] Alexander Krause, Christian Zirkelbach, and Wilhelm Hasselbring. “Simplifying Software System Monitoring through Application Discovery with ExplorViz.” In: *Symposium on Software Performance 2018: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*. Nov. 2018. URL: <http://oceanrep.geomar.de/44502/> (cited on pages 10, 68, 75, 119, 194, 235).
- [LAL15] Zengyang Li, Paris Avgeriou, and Peng Liang. “A systematic mapping study on technical debt and its management.” In: *Journal of Systems and Software* 101 (2015). DOI: <https://doi.org/10.1016/j.jss.2014.12.027>. URL: <http://www.sciencedirect.com/science/article/pii/S0164121214002854> (cited on page 60).
- [LBG+15] D. M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic. “An Empirical Study of Architectural Change in Open-Source Software Systems.” In: *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*. May 2015 (cited on page 190).
- [LC13] Miguel A Laguna and Yania Crespo. “A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring.” In: *Science of Computer Programming* 78.8 (2013) (cited on page 217).
- [Len19] Stephan Lenga. “Modernization of Monolithic Legacy Applications towards a Microservice Architecture with ExplorViz.” Master thesis. Kiel University, June 2019. URL: <http://eprints.uni-kiel.de/48007/> (cited on pages 72, 78, 121).

- [LEP+10] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. "Collaboration Tools for Global Software Engineering." In: *IEEE Software* 27.2 (2010) (cited on pages 44, 219).
- [Let12] J. Letouzey. "The sqale method for evaluating technical debt." In: *Proceedings of the Third International Workshop on Managing Technical Debt (MTD)*. June 2012. DOI: 10.1109/MTD.2012.6225997 (cited on page 60).
- [LF19] O. Levy and D. G. Feitelson. "Understanding large-scale software – a hierarchical view." In: *Proceedings of the 27th International Conference on Program Comprehension (ICPC)*. 2019 (cited on page 29).
- [Lik32] Rensis Likert. "A technique for the measurement of attitudes." In: *Archives of psychology* 22.140 (1932) (cited on pages 129, 145, 165).
- [LLS+18] D. M. Le, D. Link, A. Shahbazian, and N. Medvidovic. "An Empirical Study of Architectural Decay in Open-Source Software." In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*. Apr. 2018 (cited on page 190).
- [LS12] Qigang Liu and Xiangyang Sun. "Research of web real-time communication based on web socket." In: *International Journal of Communications, Network and System Sciences* 5.12 (2012) (cited on page 98).
- [MA10] Franco Martinig and Associates. "Methods & tools." In: *Methods & Tools: Practical knowledge for the software developer, tester and project manager* 18.1 (2010). Edited by Franco Martinig. URL: <http://www.methodsandtools.com/PDF/mt201001.pdf> (visited on 12/02/2019) (cited on page 65).
- [MBC15] Antonio Martini, Jan Bosch, and Michel Chaudron. "Investigating architectural technical debt accumulation and refactoring over time: a multiple-case study." In: *Information and Software Technology* 67 (2015). DOI: <https://doi.org/10.1016/j.infsof.2015.07.005>. URL: <http://www.sciencedirect.com/science/article/pii/S0950584915001287> (cited on page 60).

## Bibliography

- [MBN18] L. Merino, A. Bergel, and O. Nierstrasz. "Overcoming Issues of 3D Software Visualization through Immersive Augmented Reality." In: *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*. Sept. 2018. DOI: 10.1109/VISSOFT.2018.00014 (cited on page 5).
- [MC14] L. Meurice and A. Cleve. "DAHLIA: A Visual Analyzer of Database Schema Evolution." In: *Proceedings of the CSMR-WCRE*. 2014 (cited on page 216).
- [MC16] L. Meurice and A. Cleve. "DAHLIA 2.0: A Visual Analyzer of Database Usage in Dynamic and Heterogeneous Systems." In: *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*. 2016 (cited on page 216).
- [MCS+17] R. Marques, G. Costa, M. Silva, and P. Gonçalves. "A survey of failures in the software development process." In: *Proceedings of the 25th European Conference on Information Systems (ECIS)*. 2017 (cited on page 64).
- [MEH01] M. W. Maier, D. Emery, and R. Hilliard. "Software architecture: introducing IEEE Standard 1471." In: *Computer* 34.4 (2001) (cited on page 26).
- [MES+19] Hafedh Mili, Ghizlane El-Boussaidi, Anas Shatnawi, Yann-Gaël Guéhéneuc, Naouel Moha, Jean Privat, and Petko Vatlchev. *Service-Oriented Re-engineering of Legacy JEE Applications: Issues and Research Directions*. 2019. DOI: arXiv:1906.00937 (cited on page 64).
- [MFT+19] A. Mehrfard, J. Fotouhi, G. Taylor, T. Forster, N. Navab, and B. Fuerst. *A Comparative Analysis of Virtual Reality Head-Mounted Display Systems*. 2019. DOI: arXiv:1912.02913 (cited on page 97).
- [MGA+17] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz. "CityVR: Gameful Software Visualization." In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Sept. 2017. DOI: 10.1109/ICSME.2017.76 (cited on page 213).



- [MGA+18] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz. "A systematic literature review of software visualization evaluation." In: *Journal of Systems and Software* 144 (2018). DOI: <https://doi.org/10.1016/j.jss.2018.06.027>. URL: <http://www.sciencedirect.com/science/article/pii/S0164121218301237> (cited on page 209).
- [MHB+19] Leonel Merino, Mario Hess, Alexandre Bergel, Oscar Nierstrasz, and Daniel Weiskopf. "PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness." In: *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering*. 2019. DOI: [10.1145/3302541.3313104](https://doi.org/10.1145/3302541.3313104). URL: <https://doi.org/10.1145/3302541.3313104> (cited on page 215).
- [MIK+16] Anna-Liisa Mattila, Petri Ihantola, Terhi Kilamo, Antti Luoto, Mikko Nurminen, and Heli Väättäjä. "Software visualization today: systematic literature review." In: *Proceedings of the 20th International Academic Mindtrek Conference*. 2016. DOI: [10.1145/2994310.2994327](https://doi.org/10.1145/2994310.2994327). URL: <https://doi.org/10.1145/2994310.2994327> (cited on page 209).
- [MJL+15] Maurizio Martignano, Andraes Jung, Tobias Lehmann, and Christian Schmidt. "Source code analysis of flight software using a sonarqube based code quality platform." In: *Ada User Journal* 36.2 (2015) (cited on page 65).
- [MJS+00] Hausi A. Müller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley, and Kenny Wong. "Reverse engineering: a roadmap." In: *Proceedings of the Conference on The Future of Software Engineering*. 2000. DOI: [10.1145/336512.336526](https://doi.org/10.1145/336512.336526). URL: <https://doi.org/10.1145/336512.336526> (cited on page 25).
- [MLS17] Sarah Meldrum, Sherlock A. Licorish, and Bastin Tony Roy Savarimuthu. "Crowdsourced knowledge on stack overflow: a systematic mapping study." In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 2017. DOI: [10.1145/3084226.3084267](https://doi.org/10.1145/3084226.3084267). URL: <https://doi.org/10.1145/3084226.3084267> (cited on page 61).

## Bibliography

- [MLS20] L. Merino, M. Lungu, and C. Seidl. "Unleashing the potentials of immersive augmented reality for software engineering." In: *Proceedings of the IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Feb. 2020. DOI: 10.1109/SANER48275.2020.9054812 (cited on page 5).
- [MMA+20] Thaha Muhammed, Rashid Mehmood, Ehab Abozinadah, and Sanaa Sharaf. "Selecweb: a software tool for automatic selection of web frameworks." In: *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*. Edited by Rashid Mehmood, Simon See, Iyad Katib, and Imrich Chlamtác. 2020. DOI: 10.1007/978-3-030-13705-2\_14. URL: [https://doi.org/10.1007/978-3-030-13705-2\\_14](https://doi.org/10.1007/978-3-030-13705-2_14) (cited on page 64).
- [Möl17] Matthias Möller. "Experiencing Software Landscapes using HCI in ExplorViz." Bachelor thesis. Kiel University, Sept. 2017. URL: <http://eprints.uni-kiel.de/39668/> (cited on pages 14, 119).
- [MQO18] M. Majthoub, M. H. Qutqut, and Y. Odeh. "Software re-engineering: an overview." In: *Proceedings of the 8th International Conference on Computer Science and Information Technology (CSIT)*. 2018 (cited on page 24).
- [MSF+18] M. Misiak, A. Schreiber, A. Fuhrmann, S. Zur, D. Seider, and L. Nafeie. "Islandviz: a tool for visualizing modular software systems in virtual reality." In: *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*. Sept. 2018. DOI: 10.1109/VISSOFT.2018.00020 (cited on page 213).
- [MSK+19a] R. Mehra, V. S. Sharma, V. Kaulgud, and S. Podder. "Fostering positive affects in software development environments using extended reality." In: *Proceedings of the IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. May 2019. DOI: 10.1109/SEmotion.2019.00016 (cited on page 79).

- [MSK+19b] R. Mehra, V. S. Sharma, V. Kaulgud, and S. Podder. "Xrase: towards virtually tangible software using augmented reality." In: *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2019. DOI: 10.1109/ASE.2019.00135 (cited on pages 215, 221).
- [MSZ+18] M. Misiak, D. Seider, S. Zur, A. Fuhrmann, and A. Schreiber. "Immersive exploration of osgi-based software systems in virtual reality." In: *Proceedings of the IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. Mar. 2018. DOI: 10.1109/VR.2018.8446057 (cited on page 213).
- [MT05] Piyush Maheshwari and Albert Teoh. "Supporting ATAM with a collaborative Web-based software architecture evaluation tool." In: *Science of Computer Programming* 57.1 (2005). System and Software Architectures. DOI: <https://doi.org/10.1016/j.scico.2004.10.008>. URL: <http://www.sciencedirect.com/science/article/pii/S0167642304001947> (cited on page 43).
- [MT94] C. Machover and S. E. Tice. "Virtual reality." In: *IEEE Computer Graphics and Applications* 14.1 (1994) (cited on page 36).
- [Mül19] Helge Müller. "A customizable and extensible Tutorial Framework for improved Usability in ExplorViz." Master thesis. Kiel University, June 2019. URL: <http://eprints.uni-kiel.de/47038/> (cited on pages 16, 119).
- [MV95] A. Von Mayrhauser and A. M. Vans. "Program comprehension during software maintenance and evolution." In: *Computer* 28.8 (Aug. 1995). DOI: 10.1109/2.402076 (cited on page 30).
- [Mwe14] Edwin Mwendu. "Software frameworks, architectural and design patterns." In: *Journal of Software Engineering and Applications* 07 (Jan. 2014). DOI: 10.4236/jsea.2014.78061 (cited on page 64).
- [MWT95] Hausi A Müller, Kenny Wong, and Scott R Tilley. "Understanding software systems using reverse engineering technology". In: *Object-Oriented Technology for Database and Software Systems*. 1995 (cited on page 29).

## Bibliography

- [MZ15] Richard Müller and Dirk Zeckzer. “Past, Present, and Future of 3D Software Visualization - A Systematic Literature Analysis.” In: *Proceedings of the 6th International Conference on Information Visualization Theory and Applications - Volume 1: IVAPP, (VISIGRAPP 2015)*. INSTICC. 2015. DOI: 10.5220/0005325700630074 (cited on page 211).
- [NGI20] NGINX. *NGINX*. Version 1.15.8. Aug. 31, 2020. URL: <http://nginx.org> (cited on page 198).
- [NL93] Jakob Nielsen and Thomas K. Landauer. “A Mathematical Model of the Finding of Usability Problems.” In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. 1993. DOI: 10.1145/169059.169166. URL: <https://doi.org/10.1145/169059.169166> (cited on page 140).
- [NLL+18] Vijayakumar Nanjappan, Hai-Ning Liang, Feiyu Lu, Konstantinos Papangelis, Yong Yue, and Ka Lok Man. “User-elicited dual-hand interactions for manipulating 3d objects in virtual reality environments.” In: *Human-centric Computing and Information Sciences* 8.1 (2018). DOI: 10.1186/s13673-018-0154-5. URL: <https://doi.org/10.1186/s13673-018-0154-5> (cited on page 85).
- [OBr03] Michael P O’Brien. “Software Comprehension—a Review & Research Direction.” In: *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report UL-CSIS-03-3* (2003) (cited on page 30).
- [OMG15] OMG. *Unified Modeling Language*. Version 2.5. 2015 (cited on pages 5, 42, 103, 106).
- [OO00] Gary M. Olson and Judith S. Olson. “Distance matters.” In: *Human-Computer Interaction* 15.2-3 (2000). DOI: 10.1207/s15327051HCI1523\_4. URL: [https://doi.org/10.1207/s15327051HCI1523\\_4](https://doi.org/10.1207/s15327051HCI1523_4) (cited on page 44).
- [Ope20a] Open Source Software Community. *Checkstyle*. Version 8.10. Aug. 31, 2020. URL: <http://checkstyle.sourceforge.net> (cited on page 186).

## Bibliography

- [Ope20b] Open Source Software Community. *Google Web Toolkit Project (GWT)*. Version 2.8.2. Aug. 31, 2020. URL: <http://www.gwtproject.org> (cited on page 185).
- [Ope20c] Open Source Software Community. *H2*. Version 1.4.177. Aug. 31, 2020. URL: <http://www.h2database.com> (cited on page 188).
- [Ope20d] Open Source Software Community. *json:api*. Version 1.0.0. Aug. 31, 2020. URL: <https://jsonapi.org> (cited on page 193).
- [Ope20e] Open Source Software Community. *PMD*. Version 6.10.0. Aug. 31, 2020. URL: <https://pmd.github.io> (cited on page 186).
- [Ope20f] Open Source Software Community. *Sonatype*. Aug. 31, 2020. URL: <https://oss.sonatype.org> (cited on page 203).
- [Ope20g] Open Source Software Community. *Spotbugs*. Version 3.1.10. Aug. 31, 2020. URL: <https://spotbugs.github.io> (cited on page 200).
- [Ope20h] Open Source Software Community. *Swagger*. Aug. 31, 2020. URL: <https://swagger.io> (cited on page 204).
- [Ope20i] Open Source Software Community. *Traefik*. Aug. 31, 2020. URL: <https://containo.us/traefik/> (cited on page 205).
- [Ope20j] Open Source Software Community. *TravisCI*. Aug. 31, 2020. URL: <https://travis-ci.org> (cited on page 200).
- [Ora20] Oracle. *Jersey Project*. Version 2.27. Aug. 31, 2020. URL: <https://jersey.github.io> (cited on page 192).
- [PC13] Patroklos P Papapetrou and G. Ann Campbell. *Sonar in action*. 2013 (cited on page 65).
- [PDD+17] I. M. Putrama, K. T. Dermawan, G. R. Dantes, and K. Y. E. Aryanto. "Architectural evaluation of data center system using architecture tradeoff analysis method (atam): a case study." In: *Proceedings of the International Conference on Advanced Informatics, Concepts, Theory, and Applications (ICAICTA)*. Aug. 2017. DOI: 10.1109/ICAICTA.2017.8090982 (cited on pages 28, 60).
- [Pen93] David A Penny. "The software landscape: a visual formalism for programming-in-the-large." In: *PhD thesis* (1993) (cited on page 3).

## Bibliography

- [PQ+06] Marian Petre, Ed de Quincey, et al. "A gentle overview of software visualisation." In: *PPIG News Letter* (2006) (cited on pages 30, 31).
- [Pro17] Kieker Project. *Kieker user guide*. Research Report. Oct. 2017. URL: <http://eprints.uni-kiel.de/16537/> (cited on page 74).
- [PS15] A. Patidar and U. Suman. "A survey on software architecture evaluation methods." In: *Proceedings of the 2nd International Conference on Computing for Sustainable Global Development (IN-DIACom)*. Mar. 2015 (cited on pages 28, 59).
- [PS19] S. Pradeep and Y. K. Sharma. "A Pragmatic Evaluation of Stress and Performance Testing Technologies for Web Based Applications." In: *Proceedings of the Amity International Conference on Artificial Intelligence (AICAI)*. Feb. 2019. DOI: 10.1109/AICAI.2019.8701327 (cited on page 77).
- [PXW13] Claus Pahl, Huanhuan Xiong, and Ray Walshe. "A Comparison of On-Premise to Cloud Migration Approaches." In: *Service-Oriented and Cloud Computing*. Edited by Kung-Kiu Lau, Winfried Lamersdorf, and Ernesto Pimentel. 2013 (cited on page 63).
- [RCE+19a] Simone Romano, Nicola Capece, Ugo Erra, Giuseppe Scanniello, and Michele Lanza. "On the use of virtual reality in software visualization: the case of the city metaphor." In: *Information and Software Technology* 114 (2019). DOI: <https://doi.org/10.1016/j.infsof.2019.06.007>. URL: <http://www.sciencedirect.com/science/article/pii/S0950584919301405> (cited on page 213).
- [RCE+19b] Simone Romano, Nicola Capece, Ugo Erra, Giuseppe Scanniello, and Michele Lanza. "The city metaphor in software visualization: feelings, emotions, and thinking." In: *Multimedia Tools and Applications* 78.23 (2019). DOI: 10.1007/s11042-019-07748-1. URL: <https://doi.org/10.1007/s11042-019-07748-1> (cited on page 213).
- [RG08] Banani Roy and T.C. Nicholas Graham. "Methods for evaluating software architecture: a survey." In: *School of Computing TR* 545 (2008) (cited on pages 28, 59).

- [RGK18] M. Rüdél, J. Ganser, and R. Koschke. "A Controlled Experiment on Spatial Orientation in VR-Based Software Cities." In: *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*. Sept. 2018. DOI: 10.1109/VISSOFT.2018.00011 (cited on page 214).
- [RMG17] Päivi Raulamo-Jurvanen, Mika Mäntylä, and Vahid Garousi. "Choosing the right test automation tool: a grey literature review of practitioner sources." In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 2017. DOI: 10.1145/3084226.3084252. URL: <https://doi.org/10.1145/3084226.3084252> (cited on page 64).
- [RS16] Siti Rochimah and Alhaji Sheku. "Migration of existing or legacy software systems into web service-based architectures (reengineering process): a systematic literature review." In: *International Journal of Computer Applications* 133 (Jan. 2016). DOI: 10.5120/ijca2016907801 (cited on page 64).
- [RSB11] S. Ray, B. Simion, and A. Demke Brown. "Jackpine: A benchmark to evaluate spatial database performance." In: *Proceedings of the 27th International Conference on Data Engineering*. 2011. DOI: 10.1109/ICDE.2011.5767929 (cited on page 109).
- [SB99] R. van Solingen and E. Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. 1999. URL: <https://books.google.de/books?id=BuxBnwEACAAJ> (cited on pages 49, 50, 125, 183).
- [SBC+02] Rini van Solingen, Vic Basili, Gianluigi Caldiera, and H. Dieter Rombach. "Goal Question Metric (GQM) Approach." In: *Encyclopedia of Software Engineering*. 2002. DOI: 10.1002/0471028959.sof142. URL: <http://dx.doi.org/10.1002/0471028959.sof142> (cited on pages 49, 50, 125, 183).
- [SCC+02] William R Shadish, Thomas D Cook, Donald Thomas Campbell, et al. *Experimental and quasi-experimental designs for generalized causal inference*. 2002 (cited on pages 139, 140).

## Bibliography

- [Sdd+18] B. M. Santos, I. G. de Guzmán, V. V. de Camargo, M. Piattini, and C. Ebert. "Software refactoring for system modernization." In: *IEEE Software* 35.6 (2018) (cited on page 64).
- [SG16] B. Singh and S. Gautam. "The Impact of Software Development Process on Software Quality: A Review." In: *Proceedings of the 8th International Conference on Computational Intelligence and Communication Networks (CICN)*. Dec. 2016. DOI: 10.1109/CICN.2016.137 (cited on page 64).
- [SIA+17] Muhammad Shoaib, Adeed Ishaq, Muhammad Awais, Sidra Talib, Ghulam Mustafa, and Aqeel Ahmed. "Software migration frameworks for software system solutions: a systematic literature review." In: *International Journal of Advanced Computer Science and Applications* 8 (Jan. 2017). DOI: 10.14569/IJACSA.2017.081126 (cited on page 217).
- [Sie16] J. Siegmund. "Program Comprehension: Past, Present, and Future." In: *Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Volume 5. Mar. 2016. DOI: 10.1109/SANER.2016.35 (cited on pages 30, 78).
- [SKR19] 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 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. May 2019. DOI: 10.1109/ICPC.2019.00042 (cited on page 214).
- [SKR20] M. Steinbeck, R. Koschke, and M. O. Rudel. "How evostreets are observed in three-dimensional and virtual reality environments." In: *Proceedings of the IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Feb. 2020. DOI: 10.1109/SANER48275.2020.9054802 (cited on page 214).



- [SKS+17] I. Schröter, J. Krüger, J. Siegmund, and T. Leich. “Comprehending Studies on Program Comprehension.” In: *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*. 2017 (cited on page 30).
- [SMK+18] Vibhu Saujanya Sharma, Rohit Mehra, Vikrant Kaulgud, and Sanjay Podder. “An immersive future for software engineering: avenues and approaches.” In: *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. 2018. DOI: 10.1145/3183399.3183414. URL: <https://doi.org/10.1145/3183399.3183414> (cited on page 79).
- [SMK+19] V. S. Sharma, R. Mehra, V. Kaulgud, and S. Podder. “An extended reality approach for creating immersive software project workspaces.” In: *Proceedings of the IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. May 2019. DOI: 10.1109/CHASE.2019.00013 (cited on page 79).
- [SNB+19] A. Schreiber, L. Nafeie, A. Baranowski, P. Seipel, and M. Misiak. “Visualization of software architectures in virtual reality and augmented reality.” In: *Proceedings of the IEEE Aerospace Conference*. Mar. 2019. DOI: 10.1109/AERO.2019.8742198 (cited on page 213).
- [SS17] Ruslan Shaydulin and Justin Sybrandt. *To agile, or not to agile: a comparison of software development methodologies*. 2017. DOI: arXiv:1704.07469 (cited on page 64).
- [Sta17a] Gernot Starke. *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. 2017 (cited on page 26).
- [Sta17b] Miroslaw Staron. “Evaluation of automotive software architectures.” In: *Automotive Software Architectures: An Introduction*. 2017. DOI: 10.1007/978-3-319-58610-6\_6. URL: [https://doi.org/10.1007/978-3-319-58610-6\\_6](https://doi.org/10.1007/978-3-319-58610-6_6) (cited on pages 28, 60).
- [Sut65] Ivan E. Sutherland. “The Ultimate Display.” In: *Proceedings of the IFIP Congress*. 1965 (cited on page 36).

## Bibliography

- [Sut68] Ivan E Sutherland. "A head-mounted three dimensional display." In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. 1968 (cited on page 36).
- [TAV13] Edith Tom, Aybüke Aurum, and Richard Vidgen. "An exploration of technical debt." In: *Journal of Systems and Software* 86.6 (2013). DOI: <https://doi.org/10.1016/j.jss.2012.12.052>. URL: <http://www.sciencedirect.com/science/article/pii/S0164121213000022> (cited on page 25).
- [TC08] Alfredo R Teyseyre and Marcelo R Campo. "An overview of 3D software visualization." In: *IEEE transactions on visualization and computer graphics* 15.1 (2008) (cited on page 31).
- [Teu19] Daniel Teut. "Enabling Software Architecture Comparison with ExplorViz." Bachelor thesis. Kiel University, Sept. 2019. URL: <http://eprints.uni-kiel.de/47849/> (cited on pages 16, 119).
- [TJL16] Davide Taibi, Andrea Janes, and Valentina Lenarduzzi. "Towards a lean approach to reduce code smells injection: an empirical study." In: *Agile Processes, in Software Engineering, and Extreme Programming*. Edited by Helen Sharp and Tracy Hall. 2016 (cited on page 65).
- [TLP18] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. "Architectural patterns for microservices: a systematic mapping study". In: 2018 (cited on page 63).
- [TMP+19] Saulo S. de Toledo, Antonio Martini, Agata Przybyszewska, and Dag I. K. Sjøberg. "Architectural technical debt in microservices: a case study in a large company." In: *Proceedings of the Second International Conference on Technical Debt*. 2019. DOI: 10.1109/TechDebt.2019.00026. URL: <https://doi.org/10.1109/TechDebt.2019.00026> (cited on page 60).
- [TRP+17] A. Tang, M. Razavian, B. Paech, and T. Hesse. "Human Aspects in Software Architecture Decision Making: A Literature Review." In: *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*. Apr. 2017 (cited on page 190).

- [TXA+15] Leonor Teixeira, Ana Raquel Xambre, Helena Alvelos, Nelson Filipe, and Ana Luisa Ramos. "Selecting an Open-source Framework: A Practical Case Based on Software Development for Sensory Analysis." In: *Procedia Computer Science* 64 (2015). DOI: <https://doi.org/10.1016/j.procs.2015.08.525>. URL: <http://www.sciencedirect.com/science/article/pii/S1877050915026605> (cited on page 64).
- [UZX+11] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau. "Migration of SOAP-based services to RESTful services." In: *Proceedings of the 13th IEEE International Symposium on Web Systems Evolution (WSE)*. Sept. 2011 (cited on page 189).
- [VB16] L. R. Vijayarathy and C. W. Butler. "Choice of software development methodologies: do organizational, project, and team characteristics matter?" In: *IEEE Software* 33.5 (Sept. 2016). DOI: 10.1109/MS.2015.26 (cited on page 64).
- [VGC+15] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud." In: *Proceedings of the 10th Computing Colombian Conference (10CCC)*. Sept. 2015 (cited on page 217).
- [VHS+18] Christian Vögele, André Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. "WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction—a Model-Driven Approach for Session-Based Application Systems." In: *Softw. Syst. Model.* 17.2 (May 2018). DOI: 10.1007/s10270-016-0566-5. URL: <https://doi.org/10.1007/s10270-016-0566-5> (cited on page 77).
- [Vin08] Steve Vinoski. "RESTful Web Services Development Checklist." In: *IEEE Internet Computing* 12.6 (Nov. 2008) (cited on page 189).

## Bibliography

- [VNP17] J. Vincur, P. Navrat, and I. Polasek. "VR City: Software Analysis in Virtual Reality Environment." In: *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. July 2017. DOI: 10.1109/QRS-C.2017.88 (cited on page 213).
- [War19] Colin Ware. *Information Visualization : Perception for Design*. 2019. URL: <http://web.a.ebscohost.com/ehost/detail/detail?vid=0&sid=35b3830c-0855-4909-9cf4-5f6afb44c4a%40sessionmgr4010&bdata=JnNpdGU9ZWhvc3QtbGl2ZQ%3d%3d#db=nlebk&AN=114168> (cited on page 31).
- [WBR+13] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. "Analysis of the accuracy and robustness of the leap motion controller." In: *Sensors (Basel, Switzerland)* 13.5 (May 2013). DOI: 10.3390/s130506380. URL: <https://www.ncbi.nlm.nih.gov/pubmed/23673678> (cited on page 82).
- [Weg18] Josefine Wegert. "Visualizing Software Architecture Comparison of a Web-based Financial Application in ExplorViz." Master thesis. Kiel University, May 2018. URL: <http://eprints.uni-kiel.de/43099/> (cited on pages 15, 65, 72, 78, 119, 197).
- [WEH15] Jan Waller, Nils C. Ehmke, and Wilhelm Hasselbring. "Including Performance Benchmarks into Continuous Integration to Enable DevOps." In: *SIGSOFT Softw. Eng. Notes* 40.2 (Apr. 2015). DOI: 10.1145/2735399.2735416. URL: <https://doi.org/10.1145/2735399.2735416> (cited on page 235).
- [Whi07] J. Whitehead. "Collaboration in Software Engineering: A Roadmap." In: *Proceedings of Future of Software Engineering (FOSE '07)*. May 2007. DOI: 10.1109/FOSE.2007.4 (cited on pages 39, 41, 44, 219).
- [Wit16] Jan Witzany. "Instrumentierung von Android Anwendungen in ExplorViz." Bachelor thesis. Kiel University, Sept. 2016. URL: <http://eprints.uni-kiel.de/34269/> (cited on page 13).

- [WL07] R. Wettel and M. Lanza. "Visualizing Software Systems as Cities." In: *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. 2007. DOI: 10.1109/VISSOF.2007.4290706 (cited on pages 5, 32, 70, 71, 103, 108, 211).
- [WL08] Richard Wettel and Michele Lanza. "Codecity: 3d visualization of large-scale software." In: *Companion of the 30th International Conference on Software Engineering*. 2008. DOI: 10.1145/1370175.1370188. URL: <https://doi.org/10.1145/1370175.1370188> (cited on pages 32, 215).
- [WLR10] Richard Wettel, Michele Lanza, and Romain Robbes. "Empirical Validation of CodeCity: A Controlled Experiment." In: *Tech Report 2010/05* (2010) (cited on pages 33, 34).
- [WMG+10] Jim Whitehead, Ivan Mistrik, John Grundy, and Andre van der Hoek. "Collaborative software engineering: concepts and techniques." In: *Collaborative Software Engineering*. Edited by Ivan Mistrik, John Grundy, Andre Hoek, and Jim Whitehead. 2010. DOI: 10.1007/978-3-642-10294-3\_1. URL: [https://doi.org/10.1007/978-3-642-10294-3\\_1](https://doi.org/10.1007/978-3-642-10294-3_1) (cited on pages 39, 40, 42).
- [WMM19] Markus Weninger, Lukas Makor, and Hanspeter Mössenböck. "Memory leak visualization using evolving software cities." In: *Proceedings of the Symposium on Software Performance (SSP 2019)*. 2019 (cited on page 212).
- [WMM20] Markus Weninger, Lukas Makor, and Hanspeter Mössenböck. "Memory cities: visualizing heap memory evolution using the software city metaphor." In: *Proceedings of the IEEE Working Conference on Software Visualization*. 2020 (cited on page 212).
- [Woo16] E. Woods. "Software architecture in a changing world." In: *IEEE Software* 33.6 (Nov. 2016). DOI: 10.1109/MS.2016.149 (cited on page 3).
- [WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. 2012 (cited on pages 49, 126, 139, 142, 160).

## Bibliography

- [WWF+13] J. Waller, C. Wulf, F. Fittkau, P. Döhring, and W. Hasselbring. “Synchrovis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency.” In: *Proceedings of the First IEEE Working Conference on Software Visualization (VIS-SOFT)*. Sept. 2013. DOI: 10.1109/VIS-SOFT.2013.6650520 (cited on pages 5, 70, 211).
- [YAW+19] Y. Yue, I. Ahmed, Y. Wang, and D. Redmiles. “Collaboration in global software development: an investigation on research trends and evolution.” In: *Proceedings of the ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*. 2019 (cited on page 44).
- [YYO+16] Mert Yilmaz, Murat Yilmaz, Rory V. O’Connor, and Paul Clarke. “A Gamification Approach to Improve the Software Development Process by Exploring the Personality of Software Practitioners.” In: *Software Process Improvement and Capability Determination*. Edited by Paul M. Clarke, Rory V. O’Connor, Terry Rout, and Alec Dorling. 2016 (cited on page 64).
- [ZA16] Christian Zirkelbach and Marc Adolf. “An Elastic Layers Pattern Approach with Dynamically Added Layers.” In: *Proceedings of the Symposium on Software Performance 2016: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*. 2016. URL: <http://eprints.uni-kiel.de/34354/> (cited on page 77).
- [ZH19] Christian Zirkelbach and Wilhelm Hasselbring. *Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach*. Research Report 1901. Kiel University, Feb. 2019. URL: <http://eprints.uni-kiel.de/45854/> (cited on pages 2, 11, 69, 103, 121, 226, 234).
- [ZHC15] Christian Zirkelbach, Wilhelm Hasselbring, and Leslie Carr. “Combining Kieker with Gephi for Performance Analysis and Interactive Trace Visualization.” In: *Proceedings of the Symposium on Software Performance 2015: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*. 2015. URL: <http://eprints.uni-kiel.de/30101/> (cited on pages 5, 103, 121).

- [ZHF+15] Christian Zirkelbach, Wilhelm Hasselbring, Florian Fittkau, and Leslie Carr. *Performance analysis of legacy perl software via batch and interactive trace visualization*. Research Report 1509. Kiel University, Aug. 2015. URL: <http://eprints.uni-kiel.de/29430/> (cited on page 121).
- [Zir17] Christian Zirkelbach. "Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems." In: *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*. 2017. URL: <http://eprints.uni-kiel.de/37440/> (cited on pages 2, 6, 8, 68, 103, 121, 226, 234).
- [ZKH18] Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. "On the Modernization of ExplorViz towards a Microservice Architecture." In: *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*. Volume Online Proceedings for Scientific Conferences and Workshops. Feb. 2018. URL: <http://eprints.uni-kiel.de/42119/> (cited on pages 2, 10, 117, 190, 197).
- [ZKH19a] Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *Hands-On: Experiencing Software Architecture in Virtual Reality*. Research Report 1809. Kiel University, Jan. 2019. URL: <http://eprints.uni-kiel.de/45728/> (cited on pages 2, 10, 68, 80, 119, 124, 195, 210, 226, 234).
- [ZKH19b] Christian Zirkelbach, Alexander Krause, and Wilhelm 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 2019. URL: <http://eprints.uni-kiel.de/46777/> (cited on pages 2, 11, 56, 68, 74, 117, 124, 182, 210, 226, 234).
- [ZKH19c] Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. *On the Modularization of ExplorViz towards Collaborative Open Source Development*. Research Report 1902. Kiel Univer-

## Bibliography

- sity, Apr. 2019. URL: <http://eprints.uni-kiel.de/46829/> (cited on pages 2, 11, 68, 117, 124, 182, 210, 226, 234).
- [ZKH20] Christian Zirkelbach, Alexander Krause, and Wilhelm Hasselbring. "The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software." In: *International Journal on Advances in Software* vol. 13.no. 1&2 (June 2020). URL: <http://www.iariajournals.org/software/> (cited on pages 2, 12, 56, 117, 124, 182, 210, 226, 234, 235).
- [ZZ14] Jun-Feng Zhao and Jian-Tao Zhou. "Strategies and methods for cloud migration." In: *International Journal of Automation and Computing* 11.2 (2014) (cited on page 63).