

Eye Tracking Based Experiments in ExplorViz

Master's Thesis

Maria-Anna Kandsorra

May 20, 2017

KIEL UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
SOFTWARE ENGINEERING GROUP

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

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Kiel, 20. Mai 2017

Abstract

Software was never more complex than today. To improve program comprehension, several visualization tools and techniques are researched. The web-based tool ExplorViz helps developers to achieve an understanding of large software landscapes more easily than looking at source code. The communication of the landscape is visualized with live traces [Fittkau et al. 2013].

To improve a software development process and product, according to [Basili et al. 1986], are experiments necessary. Further, they help to understand, evaluate and control the process and product. Thus, the improvement to conduct experiments will help to improve a product. And in modern society and technology, it was never easier or cheaper than today to track the user's gaze. And to understand and enhance human computer interaction is an asset to be pursued.

In this thesis we present an approach to improve the optional ExplorViz experiment mode, which can create and manage questionnaires to conduct interactive experiments in ExplorViz. The approach improves the usability and enhances the experiment mode with the feature of eye tracking and recording the screen during an experiment. Further, we evaluate the approach with an experiment and confirm its functionality. The eye tracking data results do not display immediate correlations, but the data is versatile and useful.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Goals	1
1.2.1	G1: Determine Experiment Management Systems Requirements	2
1.2.2	G2: Concept and Implementation of Experiment Mode with Eye Tracking	2
1.2.3	G3: Evaluation of the Experiment Mode with Eye Tracking	2
1.3	Document Structure	2
2	Foundations and Technologies	3
2.1	ExplorViz	3
2.2	Eclipse	6
2.3	Google Web Toolkit (GWT)	6
2.4	CanJS	6
2.5	Eye tracking	7
2.6	Qt	8
2.7	JPetStore	8
2.8	Project WebRTC	8
3	Approach	9
3.1	Requirements for Experiment Management Systems	10
3.2	ExplorViz' Experiment Mode	10
3.3	Questionnaire Concept	11
3.3.1	Questiontypes	11
3.3.2	Format for Saving Questions and Answers	12
3.3.3	Management and Display	13
3.4	Concept of the Experiment Mode with Eye Tracking	15
3.4.1	Questionnaire Requirements	16
3.4.2	Eye Tracking	17
3.4.3	Screen Recording	18
4	Implementation of Questionnaire Extension	21
4.1	JSON File	22
4.2	Model	23
4.3	ExplorViz server	23
4.4	Client	23

Contents

4.4.1	ExperimentToolsPage	24
4.4.2	Option <i>Question-Interface</i>	27
4.4.3	Display during Experiment	30
5	Implementation of Eye Tracking Extension	33
5.1	JSON File	34
5.2	Server	35
5.2.1	Eye Tracking Data	35
5.2.2	Screen Recording Data	35
5.3	Client	35
5.3.1	ExperimentToolsPage	36
5.3.2	Experiment	40
6	Evaluation	47
6.1	Experiment Design	47
6.1.1	Research Questions and Hypotheses	47
6.1.2	Empirical Methods	48
6.1.3	Tasks	48
6.2	Operation	51
6.2.1	Experimental Set-up	51
6.2.2	Create Input	51
6.2.3	Tutorial	51
6.2.4	Questionnaire	51
6.2.5	Pilot Study	51
6.2.6	Procedure	52
6.3	Data Collection	52
6.3.1	Timing and Tracking Information	52
6.3.2	Correctness Information	53
6.4	Results	53
6.5	Discussion	55
6.5.1	Correctness of Tasks	55
6.5.2	Eye Tracking Data	56
6.5.3	Postquestions	60
6.6	Threads to Validity	61
6.6.1	Internal Validity	61
6.6.2	External Validity	63
6.7	Lessons Learned and Challenges Occurred	63
6.8	Summary	64
7	Related Work	65

Contents

8 Conclusions and Future Work	67
8.1 Conclusions	67
8.2 Future Work	67
Bibliography	69
Appendix A: Install Instructions	71
Appendix B: Informed Consent	73
Appendix C: Raw Experiment Results	74

List of Figures

2.1	Landscape Perspective	3
2.2	Application Perspective	4
2.3	Experiment Interface, Screenshot of ExplorViz website	5
3.1	ExplorViz Concept	9
3.2	Experiment Mode Questionnaires	12
3.3	Input Interface for Questionnaire Questions	13
3.4	Display of Prequestions	14
3.5	Display of Postquestions	15
3.6	Schematic Example of Eye Tracking Data during Experiment	16
3.7	Options for Administrator	17
4.1	ExplorViz Implementation	21
4.2	Experiment Overview RootPanel	24
4.3	Integration of CanJS	28
4.4	Extract of Question-Interface Page	29
5.1	ExplorViz Implementation	33
5.2	Screenshot of a Results Modal	37
5.3	Screenshot of a Replay Modal	37
5.4	Simple Modal before first normal Questions during Experiment	41
5.5	Security Modal when starting Screen Recording	42
5.6	Communication on Participants Computer	44
6.1	Correctness of Tasks	56
6.2	Information on the Participants Gazes	57
6.3	Duration of Experiment Participation	58
6.4	Participants Individual Correctness Information	59
6.5	Assessed Average Difficulty of Tasks	61

List of Tables

6.1	Description of the Task for the Experiment	49
6.2	Correctness Results of Participants	53
6.3	Information Ratio of Eye Tracking Data	54
6.4	Postquestions Mean and Standard Deviation	55
1	Postquestions Mean and Standard Deviation	72

Listings

2.1	Questions in TXT Format	5
2.2	Extract of an Experiment in JSON Format	6
4.1	Extract of an Experiment in JSON Format	22
4.2	Extract of ExperimentToolsPage method render	25
4.3	Extract of ExperimentToolsPage's method showExperimentList	26
4.4	Extract of ExperimentToolsPage method togglePreAndPostquestions	27
5.1	Extract of an experiment JSON file	34
5.2	Extract of an eye tracking data file	38
5.3	Extract of exp_eyeTracking Javascript file	39
5.4	Extract of eyeApi.js	43
5.5	Extract of eyejsonprotocollserializer.cpp	45
1	Command to Transform webm to mp4	71

Introduction

1.1 Motivation

In modern society and technology, to understand and enhance human computer interaction is an asset to be pursued. Considering that eye tracking for usability purposes was already used in the 1950s [Jacob and Karn 2003], it was never easier or cheaper than today to track the user's gaze.

Software was never more complex than today. To improve program comprehension, several visualization tools and techniques are researched. The web-based tool ExplorViz helps developers to achieve an understanding of large software landscapes more easily than looking at source code. The communication of the landscape is visualized with live traces [Fittkau et al. 2013]. The ExplorViz interface is interactive and allows the user to switch between two perspective levels. The landscape level shows components and their communication on different abstraction levels. On application level, ExplorViz shows a single application as an interactive 3D city model with their communications.

To improve a software development process and product, according to [Basili et al. 1986], experiments are conducted. Further, they would help to understand, evaluate and control the process and product. Thus, the improvement to conduct experiments will help us to improve a product. Tools exist for managing electronic questionnaires and automating the collection of their data, analyzing and visualizing them. An optional part of the ExplorViz tool is also an experiment mode, to create and manage interactive questionnaires to perform surveys in ExplorViz.

In this work, we want to extend the current experiment mode of ExplorViz to integrate eye tracking. The experiment mode was recently enhanced for more usability and we want to further improve it for more usability.

1.2 Goals

The main goal of this master's thesis is to enhance the ExplorViz' tool experiment mode with an eye tracking and a screen recording feature. We want to evaluate our approach and detect with empirical methods whether the eye tracking feature is a reasonable improvement for experiments with ExplorViz interactive interface. To determine this, we split our goal to three subgoals. In the following sections, we will describe them.

1. Introduction

1.2.1 G1: Determine Experiment Management Systems Requirements

Our first subgoal is to do a literature research for experiment management systems and their requirements. ExplorViz' system for managing experiments was implemented by [Finke 2014]. We want to compare the identified requirements and develop an approach to improve the experiment mode accordingly.

1.2.2 G2: Concept and Implementation of Experiment Mode with Eye Tracking

To enhance the possibilities for experiments, we want to develop an approach to enhance the ExplorViz experiment mode with eye tracking. Including identified requirements for experiment management systems, we will produce a concept which integrates with ExplorViz and implement it.

1.2.3 G3: Evaluation of the Experiment Mode with Eye Tracking

To test the implemented feature for functionality with real users, a small pilot study will be conducted. Afterwards, an experiment with the eye tracking and screen recording feature will be performed. With the results of the experiment, we will determine, whether the eye tracking and screen recording is a meaningful improvement to the experiment mode of ExplorViz.

1.3 Document Structure

This document contains the following topics. Chapter 2 describes important technologies and foundations for this thesis. Chapter 3 contains the description of our concept for improving the experiment mode and how the goals of the thesis can be achieved. Chapter 4 presents the changes we make to implement the approach for the experiment mode to achieve identified requirements. And Chapter 5 contains information about the implementation of the part of the approach to enhance the experiment mode with eye tracking and screen recording. In Chapter 6 we evaluate the changes and our approach with an experiment, also testing the functionality of the implementations and presenting the results. In Chapter 7 we present other approaches which are similar to our approach. And in Chapter 8 we will summarize our findings and results, and conclude this thesis.

Foundations and Technologies

2.1 ExplorViz

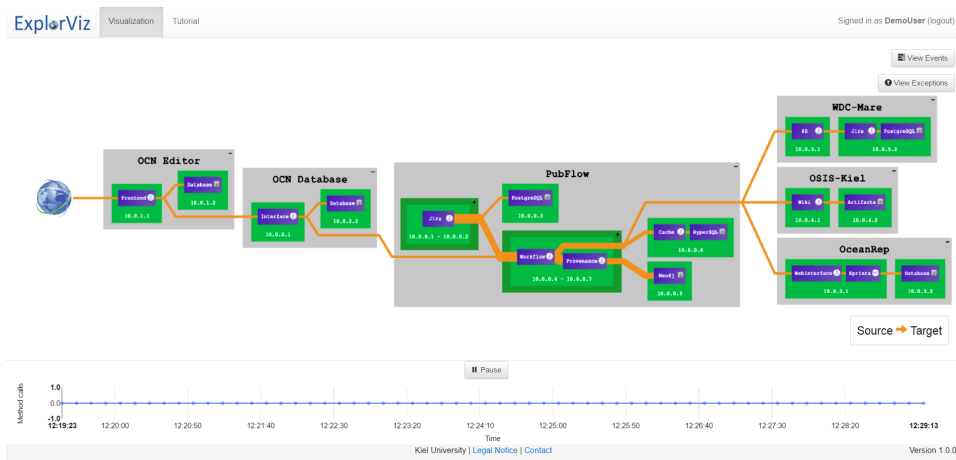


Figure 2.1. Landscape Perspective on ExplorViz Website^a

^a<http://samoa.informatik.uni-kiel.de:8181/>

ExplorViz is a web-based tool for visualizing large software landscapes [Fittkau et al. 2013]. To improve the understanding of developers for software, communication of components and their behavior are visualized in real-time based on monitoring traces. ExplorViz presents the software in two perspectives, a landscape and an application view. The software landscape view is shown in Figure 2.1, presenting components in boxes and their communication, portrayed as lines, in a flow-based layout. The boxes contain smaller boxes, a more detailed abstraction, showing their communication as well. The user can interact with them and change abstraction levels of the boxes. There is also a time-shift panel, for example to check the communication of some components at a specific time. The landscape view displays executed instances of a self-contained system, which can interact with other systems. The other perspective is shown in Figure 2.2 and displays the application level. Its components and classes and their communication are presented based on the city metaphor

2. Foundations and Technologies

[Wettel et al. 2011]. The user can interact with the components and change their abstraction level, as well as get information about the amount of communication between components. The application view represents an abstraction of the source code of the software system.

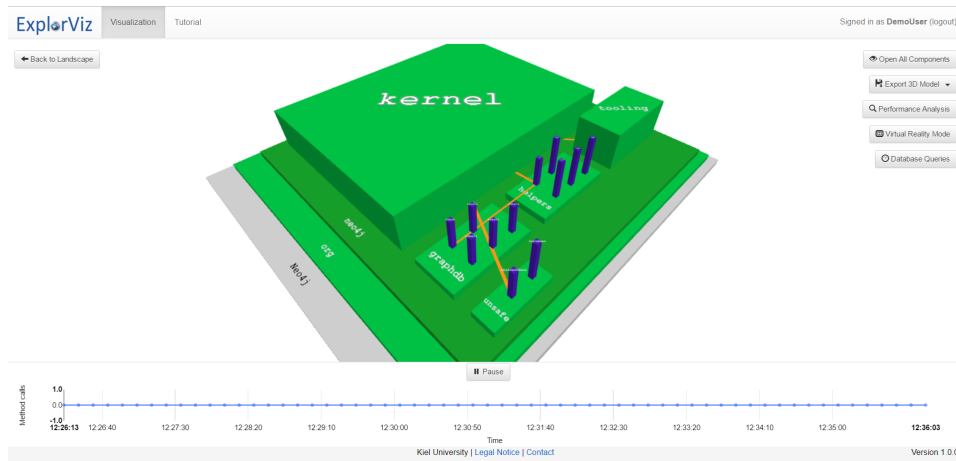


Figure 2.2. Application Perspective on ExplorViz Website^a

^a<http://samoa.informatik.uni-kiel.de:8181/>

In [Fittkau et al. 2016] it was scientifically proven that ExplorViz increases the correctness in contrast to another software visualization tool, ExTraVis [Cornelissen et al. 2007]. In comparison to ExTraVis [Cornelissen et al. 2007] it was observed by [Fittkau et al. 2016] that ExplorViz increases significantly the correct answers of participants. [Fittkau et al. 2016] used controlled experiments for this comparison as well as for different visualization variations of ExplorViz like virtual reality.

ExplorViz is implemented with GWT, Google Web Toolkit, and consists of three main parts. One is the model, containing information about classes that are important for the other two parts, the server and the client. The server holds all information and serves for the client side, giving requested information. The client side interacts through a website with user, displaying the interactive interface from Figure 2.1 and Figure 2.2. These aforementioned experiments were conducted in ExplorViz with an experiment mode by [Finke 2014] to test for usability and can also be used to evaluate new implemented features. Plain text files have to be in a specific folder on the server to be uploaded as experiments. An example can be seen in Listing 2.1. The text files can be edited via the website, are written in a custom syntax, and must be saved as 'questions.txt'. Statistical questions are also possible in the beginning of an experiment, to determine what kind of experience participants might have for example with programming languages or ExplorViz. Statistical questions afterwards can be for example their assessment of the questionnaire. The experiment mode was extended during the summer semester 2016 in the scope of a masterproject. They

2.1. ExplorViz

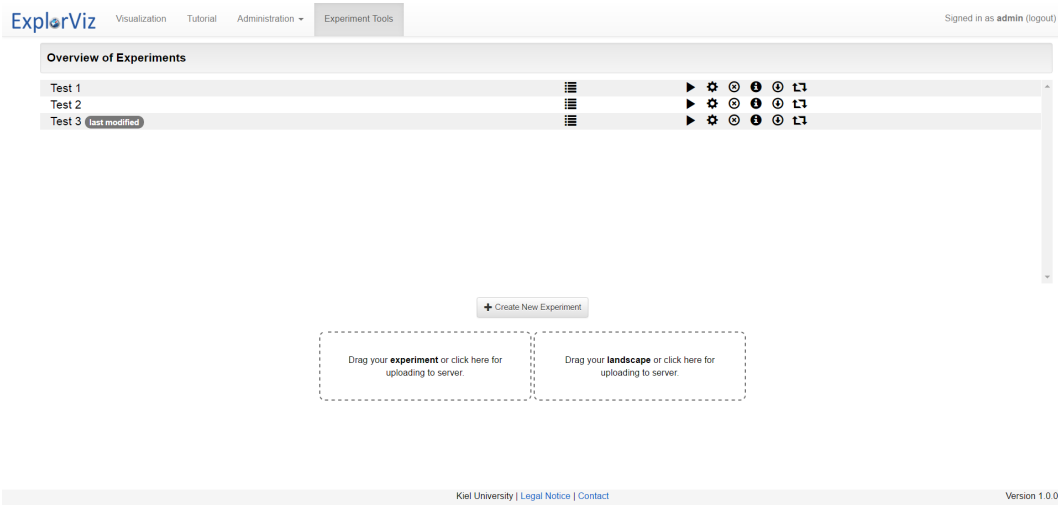


Figure 2.3. Experiment Interface, Screenshot of ExplorViz website

created a webinterface, shown in Figure 2.3. Users logged in with administrator access can create and manage new experiments. Questionnaires for specific experiments can also be created in the webinterface, with different groups in experiments and each having their own questionnaire. The experiment, its attributes and related questionnaires are saved as files in a JSON representation on the server. An example can be seen in Listing 2.2.

Listing 2.1. Questions in TXT Format

```
1 --Dialog 1 before questions, do not remove this line
2 Text: Now that you have passed the tutorial, the questionnaire will begin. Please
   give a few information about your person. Those will only be used to analyze
   the results of the experiment.
3 Number: Age
4 Placeholder: Years
5 Min: 16
6 Max: 75
7 Combobox: Gender
8 Choices: Male, Female, other
9 Tooltip:
10 Combobox: Highest completed degree
11 Choices: None, Bachelor, Master, Diploma, PhD, none of the above
12 Tooltip:
```

2. Foundations and Technologies

Listing 2.2. Extract of an Experiment in JSON Format

```
1 {
2   "questionnaires": [{
3     "questionnaireID": "quest1475325290274",
4     "questions": [{
5       "expApplication": "",
6       "answers": [
7         {
8           "answerText": "Antwort 1",
9           "checkboxChecked": false
10        },
```

2.2 Eclipse

Eclipse is an open source community, offering IDEs (integrated development environment) for different requirements. The tools can be extended with plugins and eclipse encourages teamwork and new plugins and projects. The most popular IDE is for Java developers but there are tools and plugins for nearly all programming languages. Following the instructions for developing with ExplorViz, we will use Eclipse IDE for Java and DSL Developers.

2.3 Google Web Toolkit (GWT)

GWT is used to optimize and build browser-based applications [GWT]. The developer does not need specific knowledge about the browser, XMLHttpRequest or JavaScript because Java is compiled to them, but there is also the possibility to write source code in native JavaScript. It is not only Java, but there is also the possibility to write code in Xtend, a modernized Java. Xtend allows for example type inference and is translated to Java. A drawback of GWT is its self-contained system, which makes it complicated to integrate other frameworks to it. When interacting with JavaScript objects, there must always be native JavaScript functions be implemented, which act as facilitator. ExplorViz is implemented with GWT and we will use the framework to extend the features of the experiment mode.

2.4 CanJS

CanJS is a JavaScript framework and can be used with alternative DOM libraries [CanJS]. The used elements are implemented as components which reduces dependencies and

produces more independent code. Components changed through interaction of the user can trigger rerendering parts of the interface without loading the whole page. Using templates, the amount of code is reduced and often readability is improved, instead of injecting html elements with JQuery. A part of the webinterface for managing questions of ExplorViz is already coded with CanJS. Its integration is done over GWT's native JavaScript functions, which acts among other things as facilitator between CanJS and GWT. We will extend the functionality of this implementation with the possibility of managing statistical questions, questions the user can be asked before and after the questions the experiment revolves.

2.5 Eye tracking

The first time eye tracking was used for testing usability was in 1950 [Jacob and Karn 2003]. Installing software and hardware for tracking the users eye got easier in modern times. They can be used for researching Human Computer Interaction as well as for private people, for instance for physically handicapped.

There exist different terms in the context of eye tracking, which we will define as [Sharafi et al. 2015] did. A *stimulus* is an object on the interface, that a user can see. If users look at a stimulus for more than 200ms, they fixate on a point of the stimulus and it is called a *fixation*. A *saccade* is a line of fast eye movements, staying on one point at maximum for 40-50ms. A *scan-path* is a line of fixations and saccades in a chronological order. A stimulus counts as *visited*, when it was fixated at least once by a user. These terms will be important for evaluating the eye tracking data, when the experiments with ExplorViz are performed. A common method in the industry is to generate a heat map [Sharif et al. 2016], which shows in a picture with colours which areas were visited and depending on which colour, how often the user fixated these areas. The heat map is only helpful if the interface stays fixed for the duration of the eye tracking. Interactive interfaces have to be analysed seperately, an interactive area that was often fixated holds no information on what stimuli exactly the user was looking at. In our case, this means we will have to look at each users entire eye tracking data since ExplorViz interface is interactive.

There is a Tobii EyeX eye tracker¹ available which we will use. Its SDK gives us the eye tracking data we need and currently runs only on the operating system Microsoft Windows². This restricts our extension of the ExplorViz experiment mode with eye tracking to the operating system Windows. Tobii is a swedish company, which started in 2001 as startup by three people. Today it is one of the biggest companies for producing eye tracking hardware and depending on the model, its respective software. Some small limitations we want to mention here, are the areas the user gazes at. Eye trackers are not as exact as one might naively think, this especially applies to the outer edges of the display.

¹<http://www.tobii.com/>

²<https://www.microsoft.com/de-de/windows>

2. Foundations and Technologies

2.6 Qt

We are building upon another student's work and he implemented a local server which handles the communication with the SDK of the eye tracking device. The source code runs in Qt, a cross-platform framework for software development. The programming language is C++. The executable communication server is available in the git repository³ of this thesis.

2.7 JPetStore

The JPetStore 6⁴ is a full web application built on top of MyBatis 3, Spring 4 and Stripes. It is a sample application, used to demonstrate how a web application with some classes can be built⁵. We use it as sample application for a monitoring with ExplorViz and use the created traces for the experiment for evaluating our work.

2.8 Project WebRTC

As mentioned before, we are building on top of another student's work. It uses a project called 'WebRTC-Experiment' by Muaz Khan [WebRTC-Experiment] to record the content of the screen. Currently, there exists a Chrome-Extension called *Screen Capturing*⁶, which can record the screen in cooperation with JavaScript-files and can also be saved as a local download afterwards. At the same time, the gaze of the user is recorded and saved as a local download afterwards as well. Loading these two downloads, they can be played at the same time with the gaze of the user on top of the video of the recorded screen, showing where the user gazed during the recording. Upon this Chrome-Extension, Javascript and HTML files we will base the eye tracking feature for the experiment mode in ExplorViz. Due to the chrome extension, our implementation will be restricted to the chrome browser.

³<https://build.se.informatik.uni-kiel.de/thesis/maria-anna-kandsorra-msc/tree/master/MS-Abgabe/LocalServerForEyeTracking/bin>

⁴<https://github.com/mybatis/jpetstore-6>

⁵<http://www.mybatis.org/jpetstore-6/>

⁶<https://chrome.google.com/webstore/detail/screen-capturing/ajhifddimkapgcifgcodmmfdlknahffk>

Approach

As shown in Figure 3.1, the ExplorViz project consists of three components. The server side, a client side and a model. The client and server use classes from the model, and they communicate with each other only through asynchronous remote procedure calls (RPC). The asynchronous RPC is a feature of GWT to make it easy to exchange Java objects between the server and the client. Our approach is going to modify parts in all three components.

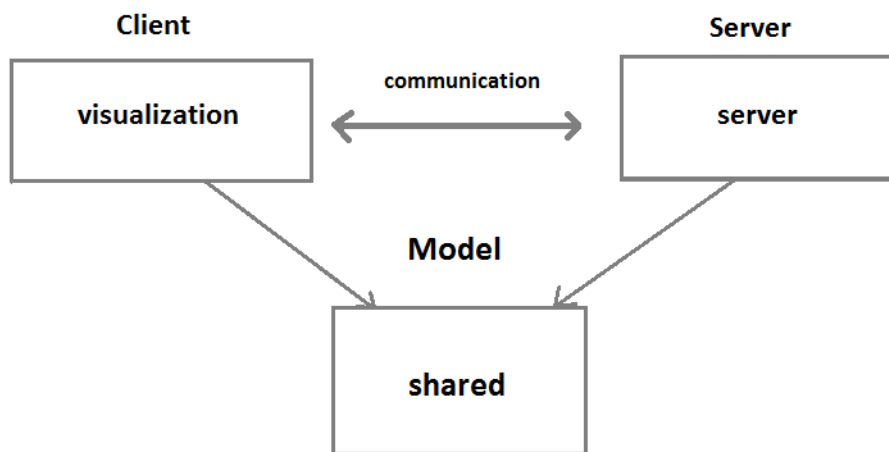


Figure 3.1. ExplorViz Concept

This chapter is about which requirements we want to fulfill with our approach and a detailed description what our approach entails. We will first take a look at requirements for experiment management systems that other scientists identified. Afterwards we will summarize what our approach is about and where we integrate it. Then we will develop our approach for the extension of ExplorViz.

3. Approach

3.1 Requirements for Experiment Management Systems

In an analysis in [Ioannidis et al. 1997], they determined three important requirements for experiment management systems. One is a uniform interface for scientists. The second is that users should not have to handle data management issues more than they need to. It should be made as simple as possible. And the last requirement is for the user to not notice the complexity of the software underneath.

[Jakobovits et al. 2000] identified even more requirements for experiment management systems. They referred to seven requirements. These requirements are originally listed with the context of experiments in the medical domain, but we can relate them to our context.

1. System Integration: The experiment management system should be applicable in many environments.
2. Data Integration: The input of the data should be the same for the user without restrictions on format or form.
3. Workflow Support: The system should keep track on who has done what on the experiments.
4. Remote Collaboration Facilities: Different parties should be able to work with the experiments and get their results.
5. Advanced Data Type Management: The experiment management system should be able to handle specific formats and types needed in the domain, as well their integrated conversion if needed.
6. Intelligent Navigation: The navigation in the experiment management system should be intuitive and constant.
7. Adaptive User Interface: Depending on the user rights, users are shown different interfaces and restrictions to access data.

3.2 ExplorViz' Experiment Mode

As mentioned in the motivation, we want to extend ExplorViz' experiment mode. ExplorViz was implemented in GWT and the code part with the experiment mode was implemented by [Finke 2014]. We will refer to it as legacy system. Of that legacy system a part of the input process for experiments was changed due to a masterproject by students in SS16. We will use the term present implementation to make it easier to refer to it.

The present experiment mode consists of a management interface for the experiments, which can only be accessed by a logged in user with an administrator role. And there exists a special interface with questions, if a user is logged in as a specific participant of an experiment, fulfilling the seventh requirement of [Jakobovits et al. 2000]. An administrator

3.3. Questionnaire Concept

also manages the user for an experiment. An experiment in ExplorViz' experiment mode in the present implementation can be a set of different questionnaires with questions. During a participation in the experiment, statistical questions are asked before and after the questions inside the questionnaire. These statistical questions can contain for example questions about the participants knowledge in a specific domain or the users age. The statistical questions after the main questions can be for example about whether the questions were easy or difficult. We want to improve the usability of this part, since the input of the data for statistical questions differs to the input of the normal questions. This contradicts the second requirement of [Jakobovits et al. 2000] and we will modify it so it is satisfied.

The details will be described in the later sections. Our approach should take the mentioned requirements in the last section into account for improving the present implementation and extending the experiment mode with eye tracking.

3.3 Questionnaire Concept

First we need to define what our future questionnaire for the ExplorViz' experiment mode should entail. There are currently the normal questions, usually why the questionnaire is done. And there are statistical questions before and after the main questions. We will refer to the questions before the main part as prequestions and to the questions after the main part as postquestions, also referring to them as questionstyles.

3.3.1 Questiontypes

In the present implementation, the normal questions offer two possible types to choose from. One is the free text type and the other is the multiple choice type. In case of the free text type, the answer of the user is put in as a text. A question with the multiple choice type gives the participant, depending on the amount of possible right answers, some text input possibilities.

The legacy system offers for the statistical questions seven possibilities for types. There is the type of number, email and input, which are nearly self exploratory. With type number, the user must put in a number, with type email, it must put in an email and with type input, the user must put in some kind of text, limited to a specific length. There is also the type binary, which lets the user choose between two possibilities. The type comment shows a bigger area for text input to the user and expects text as input and the type choice is like the type multiple choice mentioned before. There are differences in their display, but often types could be used for the same kind of statistical questions. For example could type binary be replaced by type choice, if we input just two choices. Our approach assembles all types together to just three types of pre- and postquestions. They are free text, multiple choice and number range. Free text and multiple choice comply with the types of the normal questions in the present implementation, mentioned before and lap over with the legacy types of comment and choice. They also replace binary and input. The

3. Approach

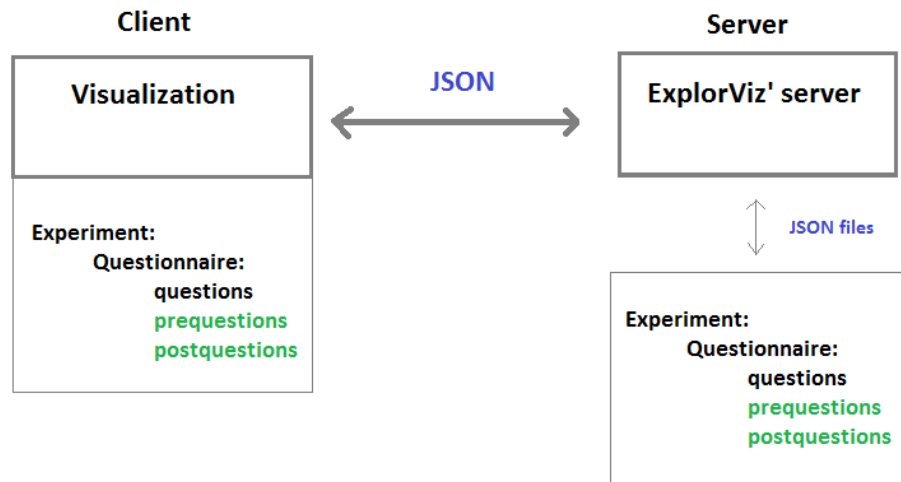


Figure 3.2. Experiment Mode Questionnaires

new type number range complies with the legacy type number. The type email was not adopted, because usually experiments and studies are done anonymously and in case that a questionnaire creator really needs the email of a user, he could let the user put it in a free text type question.

3.3.2 Format for Saving Questions and Answers

The present implementation saves the questionnaire and normal questions as part of an experiment inside a JSON file on the server. An experiment can have more than one questionnaire, see figure X, and questionnaires have questions and other attributes. The legacy system saved normal questions and saves statistical questions in text files with a self-made line by line format. We want a consistent format for the pre- and postquestions. So we choose to add prequestions and postquestions as attributes to a questionnaire and save them as well inside the JSON file like the present implementation. In Figure 3.2 we can see the questionstyles on both communication sides colored in green.

The answers are currently saved in a CSV file after every submission of the answer. Together with the answers for the normal questions are the taken time in milliseconds, the time when the question was started and ended in milliseconds and the id of the user. The answers for statistical questions are split in pre- and postquestions. Each have one line with all answers and the userId of the participant. This is needed to interpret the data with R.

3.3. Questionnaire Concept

The image shows a web browser window with the address bar displaying 'http://explorviz.net'. The browser has three tabs: 'Visualization', 'Tutorial', and 'Experiment Tools'. The main content area features a 'Question Interface' panel. This panel includes a 'Type' dropdown menu currently set to 'Multiple-Choice', a 'Question Text' text input field with the placeholder '<<QuestionText>>', and a 'Possible answers' section with a text input field containing the placeholder '<<answer 1>>'. Below the input fields are several control buttons: a red 'Delete' button, a 'Pre- Questions Post-' button, and a 'Back<<' button. At the bottom of the panel are '>>Save & Forward' and 'Exit' buttons.

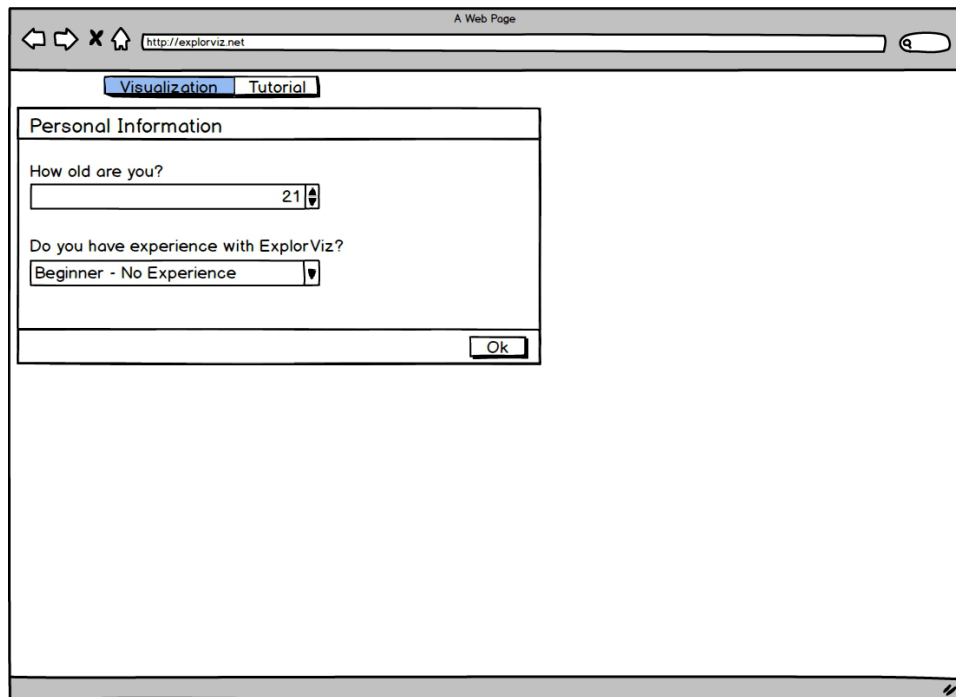
Figure 3.3. Input Interface for Questionnaire Questions

3.3.3 Management and Display

For our defined questionnaire, there are two points really important in this context. One is, how to create, delete, and manage the questionnaire from an administrator's view. And the second point is, how a participant experiences doing the questionnaire. The legacy system of course supports these points, as well as the present implementation. In the present implementation the usability of the first point for normal questions was increased.

The present implementation entails a webinterface for the administrator, to modify, load and create new experiments. There is a possibility to drag and drop experiment- and landscapefiles into the webinterface, to upload and save them. There is also a user management, for creating, deleting and printing users out. Continuing on, to modify, delete and add questions to the questionnaire, another interface was implemented. In the legacy code base, textfiles had to be edited inside a basic interface, with a specific line by line format. In the present implementation a new site is opened and next to a landscape, a slideable interface gives the administrator the option to modify questions. To have a consistent and intuitive design, an interface for the pre- and postquestions should be as close to the mentioned slider as possible and add buttons to switch between the question styles, see Figure 3.3. We do this also for conforming to the requirement of [Ioannidis et al.

3. Approach



The image shows a web browser window titled "A Web Page" with the address bar containing "http://explorviz.net". The page has two tabs: "Visualization" (selected) and "Tutorial". A modal form titled "Personal Information" is displayed. It contains the following elements:

- A text input field for "How old are you?" with the value "21" and a spinner control.
- A dropdown menu for "Do you have experience with ExplorViz?" with the selected option "Beginner - No Experience".
- An "Ok" button at the bottom right of the form.

Figure 3.4. Display of Prequestions

1997]. Further, the person creating the questionnaire should be able to either turn on or off the option of pre- and postquestions. If during the input process, pre- or postquestions were not added, they will of course not be displayed and be skipped. They would also be skipped, if there were any pre- or postquestions created but the option was turned off. Like that, the possibility of only prequestions or only postquestions would be possible. The present implementation changed the process to save and load the main questions as attributes of the questionnaire. We will update this process for the pre- and postquestions as well.

Now to the before mentioned second point, how a participant experiences the questionnaire. This part was not changed from a participants view in the present implementation. It is still the same from the legacy system. At first all statistical prequestions are divided into two modals, which are shown after each other. Then the main part with the normal questions starts, where each question has their own modal. The statistical questions afterwards are again shown in two modals. We want to change that and display all prequestions at once inside one modal and do the same with all postquestions, see Figure 3.4 and Figure 3.5. The user has the possibility to overview all prequestions at the beginning and there is no confusion about when the normal questions start. Also, this way the user can intuitively differentiate between the prequestions and the normal questions, that there

3.4. Concept of the Experiment Mode with Eye Tracking

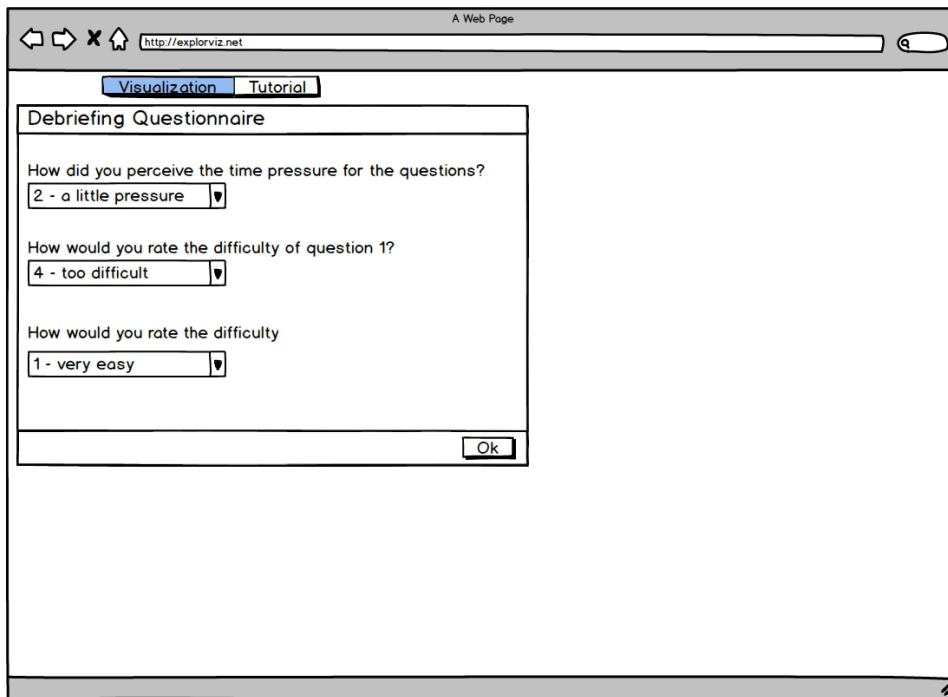


Figure 3.5. Display of Postquestions

lays more importance to the normal questions than to the prequestions. The same goes afterwards to the postquestions. The user might be impatient to finish the questionnaire after he answered many normal questions. When all postquestions are shown at once, the user is able to overview them and can guess, that the questionnaire is finished afterwards. We make it more transparent for users to see how many questions are left to answer.

3.4 Concept of the Experiment Mode with Eye Tracking

To have the eye tracking data of the eye tracker alone, is not very useful. We know when and where on the display the user gazed, but we do not know what was displayed, for example see Figure 3.6. So a recording of the time the eyes are tracked is needed. The following section will be about the requirements for the questionnaire and then the eye tracking part. Afterwards we will talk about how we will record the screen during the time we track the users eyes.

3. Approach

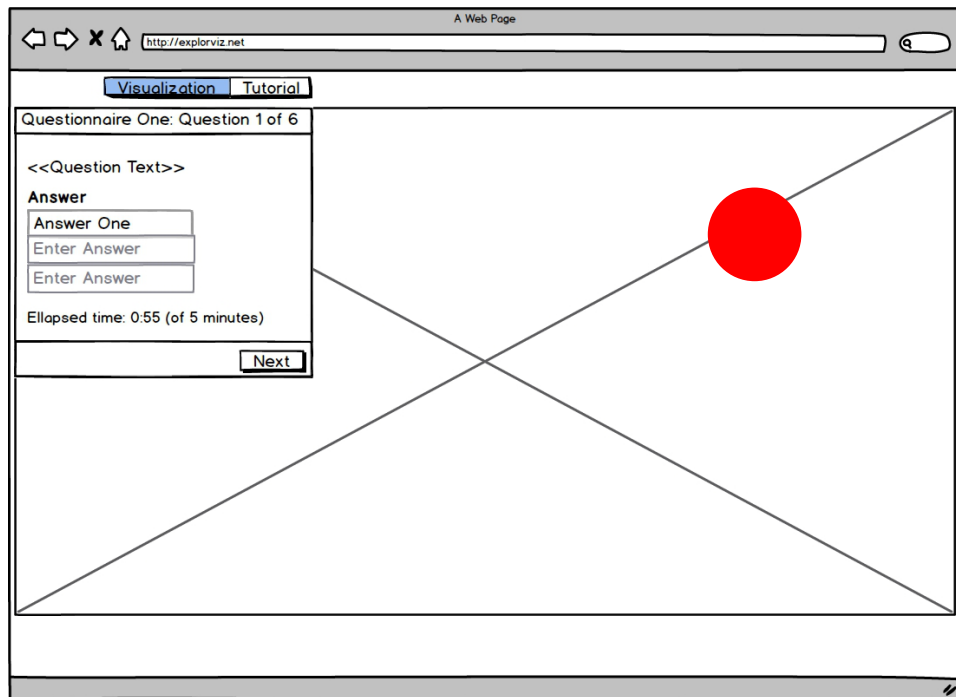


Figure 3.6. Schematic Example of Eye Tracking Data during Experiment

3.4.1 Questionnaire Requirements

As an administrator creating and managing questionnaires, it should be possible to turn these two options on and off individually, because one might not have an eye tracker but still want to record the screen of a user during the participation of an experiment. The other way is not possible, because only the eye tracking data without the recording of the screen would not be useful. The default should be off for both, because they are special options which need specific hardware and software and someone who only wants to create a questionnaire would not want a special option turned on. As mentioned, these options can be turned on or off for every questionnaire, and the other questionnaires of the experiment are left untouched. A mockup design is shown in Figure 3.7. It should be possible to play the recording after a user participated in an experiment. The administrator should be able to select a user's recording. This can be achieved through a modal showing information about all users of the questionnaire. This information should be about whether a user finished the questionnaire, if there exists any eye tracking data, or screen recording. Also, there should be a possibility for downloading the user's information after ending a questionnaire. From this modal, the administrator should be able to select a user and watch his screen recording. Before and during the playing of the recording, the watcher should also be able

3.4. Concept of the Experiment Mode with Eye Tracking

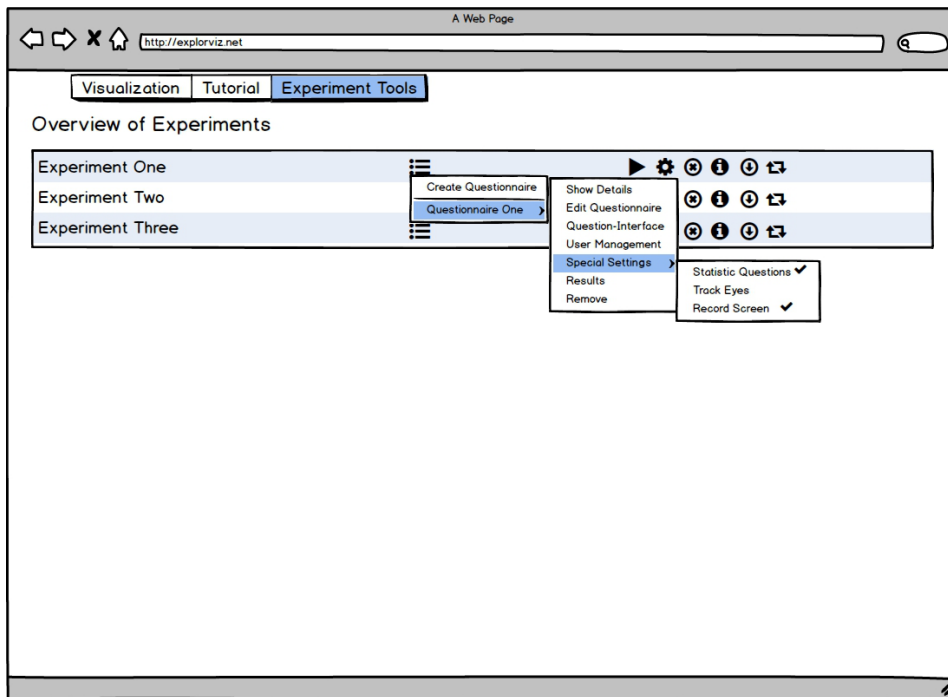


Figure 3.7. Options for Administrator

to turn the displaying of the eye tracking data on or off in the recording. So they can watch, where the user gazed during the recording.

3.4.2 Eye Tracking

To track the eyes, we should take four points into account. What is the eye tracking data, how do we get the eye tracking data, what do we do with the eye tracking data and what about calibration of the eye tracker.

Eye Tracking Data Format

What the eye tracking data consists of, is dictated by the SDK of the eye tracker, but we need the x and y position of the point on the display where the user gazed and when the eye tracker recorded that. Also, for the sake of displaying afterwards where the user gazed during the recording, we need to save the width and height of the screen where the recording took place. This can be handled by the common format JSON.

3. Approach

Get Eye Tracking Data

To get the eye tracking data, we need an eye tracker and communicate with it during the participation of the questionnaire. In our case, we restrict it to the normal questions, where the participant interacts with the ExplorViz interface. So before starting the main part, we have to start the communication with the hardware and end it after the last normal question is answered. We need a SDK for the eye tracker and its software, in our case we will use an external server which communicates with the hardware and then sends the eye tracking data to the questionnaire website. This will limit the first requirement, System Integration, of [Jakobovits et al. 2000].

Eye Tracking Data at the Client

At the clients side, we save it first in an array and save the array afterwards in a more permanent form. In our case, we upload it to the server of ExplorViz where it is saved in the mentioned format, JSON, as a file.

Calibration of Hardware

Usually, this has to be done for every participant. Depending on the SDK and eye tracker, the calibration can be implemented by us before the participation in an experiment. Or, if this is not possible with the SDK, the eye tracker offers software to do this. The calibration should take place as close as possible to the actual recording, so it is as accurate as possible but without disturbing the questionnaire flow. If we calibrate for instance directly after the prequestions and before the main part, the normal questions, the pressure at first might be high and the user might act and look differently as usual.

3.4.3 Screen Recording

The recording of the screen should take place as close as possible to the start and stop of the recording of the eyes. Depending on the implementation, to start and stop in parallel would be optimal, but in a one threaded browser, this is not possible. Further, the recording should be in a common format and uploaded and saved on a server. The last mentioned part is really important to make it possible to watch the video together with the eye tracking data afterwards. To reuse already written code, we can use an open source library which records the screen with the help of an extension in the browser, since the experiment is done in a browser. But there are drawbacks. For security reasons, the browser needs the explicit permission of the participant to record the screen, which interrupts the flow of the experiment. Also, we have to mind memory restrictions of the browser because video media is usually relatively big in contrast to text and images, which are more commonly used in websites. Alternatively, we could use a desktop software, but in this case we have to record the screen from the beginning until the end of the full experiment, which leaves too much footage that we do not need. Another drawback is the software must be first

3.4. Concept of the Experiment Mode with Eye Tracking

downloaded and then be installed by the user, and further the recording must be uploaded by the participant manually. We want to make it as simple as possible for participants, even if we have to work with the mentioned drawbacks, so we decide for the recording of the screen via a library.

Implementation of Questionnaire Extension

In section 3.3 we developed our approach to extend the questionnaire inside the experiment mode of ExplorViz so it will conform more to the requirements of Experiment Management Systems. We describe how we modified the current ExplorViz project at different parts to add the questionstyles *prequestion* and *postquestion* to the Questionnaire.

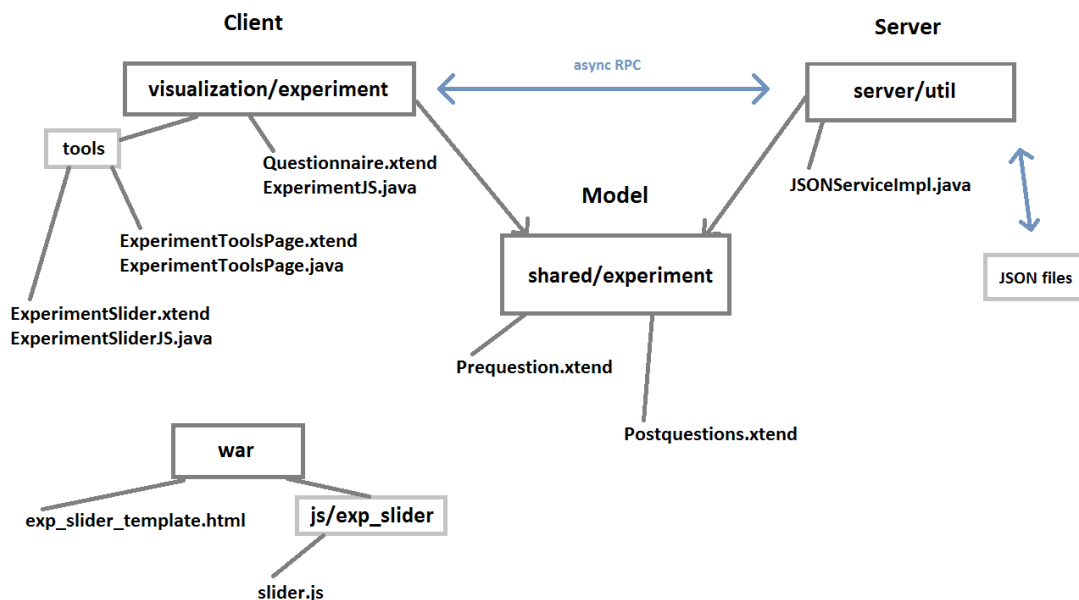


Figure 4.1. ExplorViz Implementation

Figure 4.1 shows a more detailed overview of the components of the implementation of ExplorViz and the classes that exist and we will implement. The names inside the boxes comply with the package names of the project, except for the JSON files. The names underneath the boxes, connected by lines, are classes we modify and in case of the *Model*

4. Implementation of Questionnaire Extension

part, create. The blue lines represent a communication between components or in case of the JSON files the access of them. The package *war* is not in the source code but is the static files of the website. In the following sections, we describe the illustrated and their relationships in detail.

4.1 JSON File

Listing 4.1. Extract of an Experiment in JSON Format

```
1 {
2   "questionnaires": [{
3     "preAndPostquestions": true,
4     "questionnaireID": "quest1493635916790",
5     "prequestions": [{
6       "expApplication": "",
7       "answers": [],
8       "workingTime": "",
9       "type": "freeText",
10      "questionText": "Prequestion One"
11    }],
12    "recordScreen": false,
13    "questions": [{
14      "expApplication": "",
15      "answers": [
16        {
17          "answerText": "Answer One",
18          "checkboxChecked": false
19        },
20        {
21          "answerText": "Answer Two",
22          "checkboxChecked": false
23        }
24      ],
25      "workingTime": "5",
26      "type": "freeText",
27      "expLandscape": "1467188123864-6247035",
28      "questionText": "Question One"
29    }],
```

As shown in Listing 4.1, the attribute `questionnaire` is extended by two additional lists for questions. They are called *prequestions* and *postquestions*. Important is also the attribute

preAndPostquestions inside every questionnaire. This determines the option whether pre- and postquestions should be enabled during the input of the questions and during an experiment.

4.2 Model

The model side of ExplorViz project is named *shared* and contains classes which are used on the client and the server side of the project. This circumstance is based on the characteristics of GWT, that the client side is implemented in Java. Before, in the legacy code, there was a class named *StatisticalQuestion*. We replace this class with two other classes, the *Prequestion* and the *Postquestion*. *StatisticalQuestion* assign a type to each question, for example *number* and depending on this, the inserted HTML will change for the representation of the question inside the questionnaire-modal during an experiment. Important attributes like answers, the text of a question, and ids are part of the class as well.

Prequestion possesses these attributes as well, inheriting from already existing class *Question* and we add the integer attributes *min* and *max*, for the new type *number range* mentioned in the approach. We also add the functionality for inserting the correct HTML for the representation of the new types mentioned in the approach, *free text*, *multiple choice*, and *number range*.

4.3 ExplorViz server

On the server side of the architecture of ExplorViz, which is illustrated as *server*. As shown in Figure 4.1, the interface *JSONServiceImpl* provides operations to handle experiment files, which are stored in the JSON format. This means, we implement functions to access the JSON files to modify the questionnaire attribute *preAndPostquestions*. There is no need for a set-function of the list attributes *prequestions* and *postquestions*, because they get set when an experiment as a whole gets saved or modified. In that case a JSON string is sent as parameter and saved in the respective file as a whole, without modifying or filtering the content, except validating it as JSON and match it to a prefabricated experiment schema.

4.4 Client

As seen in Figure 4.1 and mentioned before, ExplorViz project has a client side, called *visualization*. In the following, we describe the relevant parts on the side of the client, which need to be changed or developed.

4. Implementation of Questionnaire Extension

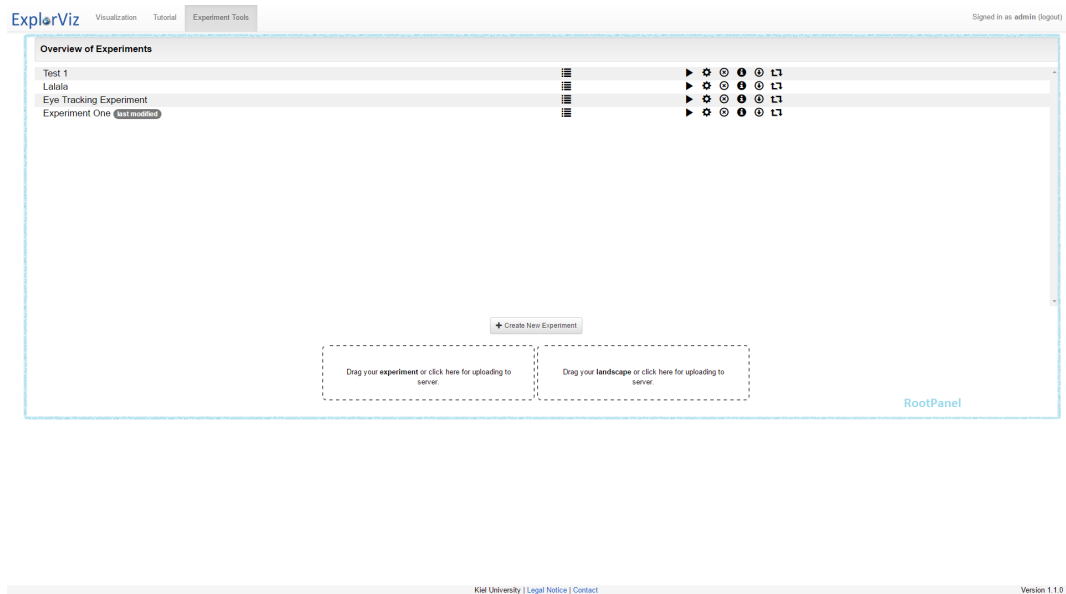


Figure 4.2. Experiment Overview RootPanel

4.4.1 ExperimentToolsPage

The class *ExperimentToolsPage.xtend* and its counterpart *ExperimentToolsPageJS.java*, see Figure 4.1, create the administrator experiment overview page shown in a screenshot in Figure 4.2. We want to modify this existing interface and add the option of pre- and postquestions. Since that name is long, we name it *Statistic Questions*. Based on the mockup as shown in Figure 3.6, we add a menu option *Special Settings*, add the menu option *Statistic Questions*, and show with a glyphicon, a small picture of a black tick, whether the option is enabled or disabled. In the following sections we will also explain some important principles of the ExplorViz project implementation and its used features of GWT.

Add Option Statistic Questions

To add the two menu options *Special Settings* and *Statistic Questions*, we need to add their HTML representation into their respective places. To describe where we add their representation, we need to explain how the administrators overview is created. GWT gives the developer the possibility to use Java to implement the web pages. One of the classes to act as a HTML element, is *RootPanel*. When compiled to HTML and Javascript, the *RootPanel* is the first container inside the body after a navigation bar, as shown in Figure 4.2 with an indication where *RootPanel* is located. If we set the *RootPanel*'s view with a string containing valid HTML, we set the content of the page. To do this, there

4.4. Client

is a big HTML string in class *ExperimentToolsPage* in method *showExperimentList*. During the loading of the page, the *ExperimentToolsPage* first requests the experiments from the *ExplorViz* server, see Listing 4.3. They are sent back as string, containing a JSON string and given to the function *showExperimentList* which gets executed afterwards. The communication between *ExperimentToolsPage* and the server is done via asynchronous RPC. As mentioned before, this kind of communication is a feature of GWT to make it easy to exchange Java objects. When the RPC returns, a callback is invoked. In this case, the *GenericFuncCallback* is called and it invokes the function we pass on to it, as shown in Listing 4.3, the function *showExperimentList*. The callback also needs the type of the return-value by the server, to pass it on as parameter to the function, in Listing 4.3 it is the common type of *String*.

Listing 4.2. Extract of *ExperimentToolsPage* method *render*

```
1 jsonService.getExperimentTitlesAndFileNames(  
2     new GenericFuncCallback<String>([showExperimentList])  
3     )
```

With the information about the experiments as parameter, the HTML for the content of the page is built and set inside the function *showExperimentList*. This string containing HTML is the part we add the menu options. See Listing 4.4 with the specific code part we add. Due to Xtend and the source code generation, we can generate the string. The code extract in Listing 4.4 does not show it, but it is located inside two for-loops of code generation. We see an indication first in line 13, where the expression *«i.toString + j.toString»* is shown. The variables *i* and *j* are the counters of each for-loop. They indicate how many experiments and questionnaires we went through, making the id of the link unique.

4. Implementation of Questionnaire Extension

Listing 4.3. Extract of ExperimentToolsPage's method showExperimentList

```
1  ...
2      <li><a id="expUserManQuestSpan«i.toString + j.toString»">User
3          Management</a></li>
4
5      <li class="dropdown-submenu">
6          <a style="word-wrap: break-word; white-space: normal;">Special
7              Settings</a>
8          <ul class="dropdown-menu" >
9              <li><a id="expEditStatQuestionsSpan«i.toString + j.toString»">
10                 Statistic Questions
11                 «IF questionnaire.getBoolean("preAndPostquestions") == true»
12                 <span class="glyphicon glyphicon-ok" title="Statistical
13                     Questions is true"></span>
14                 «ENDIF»
15             </a></li>
16         </ul>
17     </li>
```

ClickHandler

As shown in Listing 4.4, the tick glyphicon is displayed when the option *preAndPostquestions* is true inside the respective questionnaire. We want to add functionality to this menu option, to enable and disable the option *Statistic Questions*, the *preAndPostquestion* attribute inside a saved questionnaire. Considering that, we need a clickHandler which on the one hand sets the attribute to its contrary and shows a success modal after successfully changing the attribute. And on the other hand toggles the display of the tick glyphicon. The clickHandler gets assigned through the id of the button. At first we check whether an experiment is running, and in that case do not let the user change anything. Then we request the boolean value of the attribute *preAndPostquestions* in the questionnaire the user clicked on. Afterwards, because of the asynchronous request to the ExplorViz server, another function named *togglePreAndPostquestions* is called. As shown in Listing 4.5, *togglePreAndPostquestions* toggles the boolean value and calls a function which starts the process to let the glyphicon vanish or appear in the respective questionnaire option. And *togglePreAndPostquestions* updates the value on the server as well as calls a function which shows a modal which informs the user that the toggling of the option was successful.

Listing 4.4. Extract of ExperimentToolsPage method togglePreAndPostquestions

```

1  def static togglePreAndPostquestions(String experimentFileName, String
    questionnaireID, boolean serverPreAndPostquestions) {
2      //toggle and update preAndPostquestions
3      var preAndPostquestions = !serverPreAndPostquestions;
4      toggleGlyphicon(serverPreAndPostquestions, "expEditStatQuestionsSpan",
        experimentFileName, questionnaireID);
5      jsonService.setQuestionnairePreAndPostquestions(experimentFileName,
        questionnaireID, preAndPostquestions, new GenericFuncCallback<Void>([]))
        ;
6      ExperimentToolsPageJS::showSuccessMessage("Option Statistical Questions", "
        The option for pre- and postquestions was set to " + preAndPostquestions
        .toString() + ".")
7  }

```

Toggling the Glyphicon

We are now inside the function *toggleGlyphicon*, that we called in line 4 of Listing 4.5. To toggle the tick glyphicon in the menu option, we need to know the id of the menu option. But we only have the experiment name and the questionnaireId as context. We start with requesting all experiments and their questionnaire, which we go through in the search for the correct experiment name and then the correct questionnaireId. Then we call a function which is implemented with native Javascript, named the same as our current function: *toggleGlyphicon*. These kind of native Javascript functions exist in the ExplorViz project quite often and are implemented inside a counterpart Java class, in this case called *ExperimentToolsPageJS.java*. The functions are usually public, static, and because of the native Javascript are they also tagged *native*, and are called JavaScript Native Interface¹ (JSNI) methods. And here the native function *toggleGlyphicon* either removes or appends the tick glyphicon with the help of JQuery. It is noteworthy that it is called native Javascript, but that there are differences to normal Javascript. One of them is, that we need to add *\$wnd* to access Javascript instances of the page. Also, and this is a big advantage, we can call Java methods and pass on variables as parameters.

4.4.2 Option Question-Interface

The administrator has more possibilities than just to change an option of a questionnaire. An important part is to add questions. It was already mentioned that there exists an interface to add, modify and delete questions of a questionnaire. But as mentioned before, we added the pre- and postquestions to the questionnaire. And in the approach, we want

¹<http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSNI.html>

4. Implementation of Questionnaire Extension

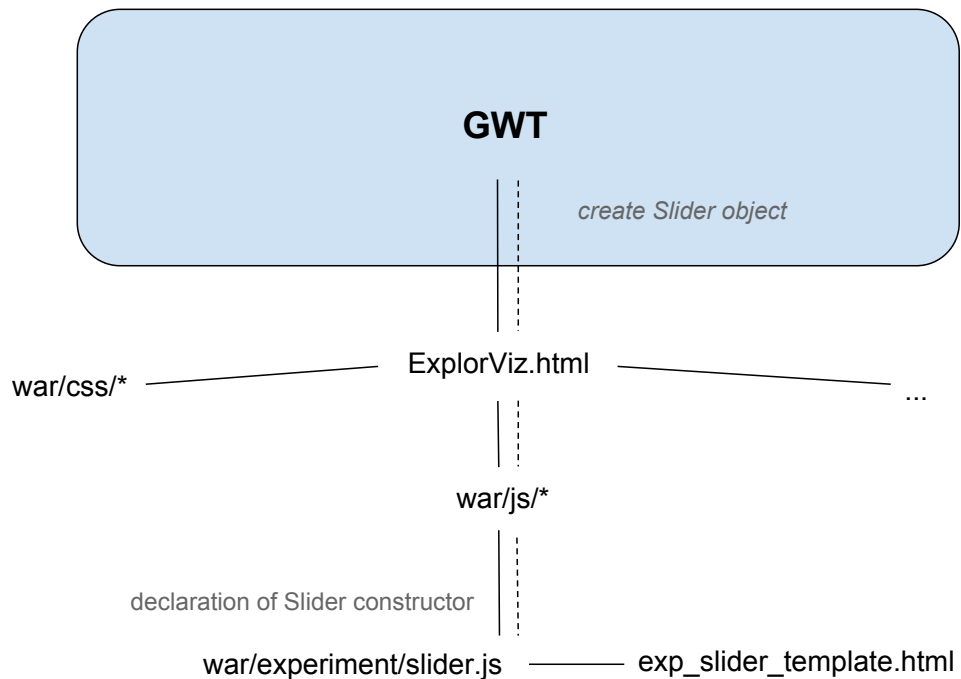


Figure 4.3. Integration of CanJS

to add the possibility of managing them inside the already existing interface, for consistent design. The interface was implemented with CanJS and we implement our approach there as well.

The menu option *Question-Interface*, shown in the approach mockup Figure 3.7, is implemented like the menu option before. As HTML added to the view in *ExperimentToolsPage* and combined with a *clickHandler*. But this *clickHandler* changes to a page called *ExperimentSlider*. This Xtend class implements specific methods and requests and transfers respective questionnaire data to a Javascript object. In Javascript a function is an object at the same time and the mentioned Javascript object is named *Slider*. As shown in Figure 4.3, the connections here exists through the main HTML file, *ExplorViz.html*. It includes various Javascript libraries and also the *slider.js*. The call and passing on of the data happens inside the counterpart class of *ExperimentSlider*, *ExperimentSliderJS.java*, inside the blue GWT box in Figure 4.3, illustrated with light blue. The *Slider* receives for parameters questionnaire data but also functions, which are implemented in the class *ExperimentSlider*. As stated before, we can invoke Java functions from inside the native JavaScript. Therefore, we create functions inside the *ExperimentSliderJS* native JavaScript and pass them on as parameters

4.4. Client

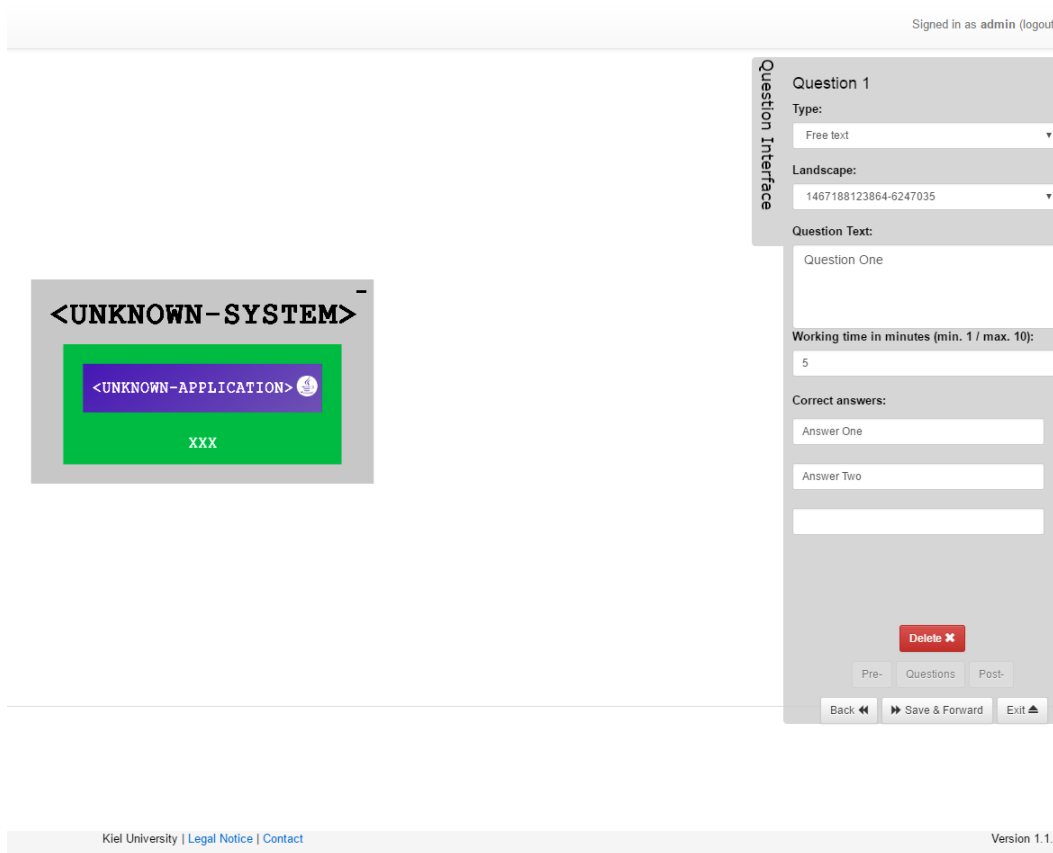


Figure 4.4. Extract of Question-Interface Page

to the *external* JavaScript object *Slider*, as illustrated as light blue line in Figure 4.3. As a result, the *Slider* outside of the GWT environment can use the functionality and context variables of class *ExperimentSlider* inside GWT. The functions that are given as parameter entail to save data on the *ExplorViz* server, get the last viewed landscape, to load a specific landscape, and a function to get back to *ExperimentToolsPage*, the *Experiment Overview* page.

CanJS interacts via components, which can be parts of each other. A component consists of a template containing HTML, variables and events. And they are named via a *tag*. We can access values inside the template with variables declared in the component. Also noteworthy is, that if the variables are changed, a rerendering of the component gets triggered.

To add functionality to make questions of the styles *prequestions* or *postquestions*, we must give the *Slider* the information whether *Statistic Questions* are enabled or not. We

4. Implementation of Questionnaire Extension

do that with a boolean parameter called *preAndPostQuestions*. Inside the root components template of Slider, we check for the value *preAndPostQuestions* and either disable or enable a components with a group of buttons. As shown is an example screenshot in Figure 4.4, the button series with *Pre-*, *Questions*, and *Post-* are disabled. When they are disabled, the Slider shows only normal questions. When the option is enabled, it follows that the buttons are enabled, except the one in which context the administrator is. That button is disabled and the questions of that question style are shown. During a switch of the questionstyle context, the data gets saved if it is valid. Modals pop up if questions are not valid, and the invalid questions gets discarded if the modals are accepted. Due to CanJS, we only need to change a variable inside the root component, and the Slider gets updated. Important to note here is, that we also need to create a new component for the input in case of the pre- or postquestiontype of *Number Range*. Therefore we add a template and append the component at their respective places inside the code. Therefore whenever the current questiontype *Number Range* is, to change the root components input to the one of *Number Range*. The new component contains two number inputs, a minimum and a maximum, to input a restriction for the range of numbers experiment participants can input.

Inside the class *ExperimentSliderJS* exists also the implementation for a tour an administrator gets shown when he opens the Question-Interface page for the first time. The Question-Interface page is manipulated by native functions to show small text modals on specific elements inside the Slider. Since we added a button group, we update the tour with some information.

4.4.3 Display during Experiment

With the sections before, we modified the possibilities for the questionnaire input and its saving process. Hereafter, we describe the editing of the questionnaire display during an experiment participation.

When a participant logs into ExplorViz, the web page recognizes him as such and refers him first to the tutorial. The page class is called *Experiment.xtend* and there exists the counterpart class *ExperimentJS.java*. The Experiment class handles mostly the tutorial, its steps and the loading of its texts. When the tutorial is completed or if the participant clicks on the navigation bar link *Visualization*, the class *Questionnaire* instead acts as Experiment, also calling methods from *ExperimentJS.java*.

The class *Questionnaire* starts with requesting the questionnaire attribute *preAndPostquestions*, to determine whether the experiment process should start with the prequestions or skip them and continue with the normal questions. The request happens, as mentioned before, with an asynchronous RPC to the server. We request the experiment name and the normal questions from the server and save them as attribute of class *Questionnaire*. Afterwards we start a process to begin the experiment visually.

A modal that informs the participant that he takes part in an experiment gets injected into the page, showing the name of the experiment. When the participant confirms, either all

prequestions or the first normal question gets shown in the modal. In case of prequestions, is the attribute *preAndPostquestions* of the class *Questionnaire* not *false*. We request the prequestions from the server and create a form containing them in HTML. This form is inserted into the modal. The form gets created with the help of the methods of *Model* package *shared.experiment*, remember Figure 4.1, class *Prequestion*. Depending on the type of *Prequestion*, the HTML code elements get added to the form. When the participant answered all prequestions and confirmed the modal, the first normal question gets injected into the modal. After the confirmation and before the injecting of the questions, the answers of the participant to the prequestions get saved. We can reuse the old function to save the answers, since the format of the HTML form of the prequestions is the same as before.

The normal questions get shown one after the other, the time the participant takes is tracked and also shown inside the modal. Also, the respective software landscape for the question gets loaded. After each question, the answer is saved and the next questions displayed. This was implemented by [Finke 2014], as mentioned before. We do not modify anything here.

After the last normal question, the visualization of the software landscape is emptied and checked whether *preAndPostquestions* for the *Questionnaire* is true. If it is false, the question modal gets closed and the participant gets logged out and the user login is set to done. If the postquestions are to be displayed, they are first requested from the server and then put into a form, like the prequestions before. And after the input and confirmation in the modal, the answers also get saved, the participant gets logged out and set to done.

Implementation of Eye Tracking Extension

In section 3.4 we modeled our approach to track the eyes of a participant during an experiment. We explain here, what we modify in the ExplorViz project to implement our approach.

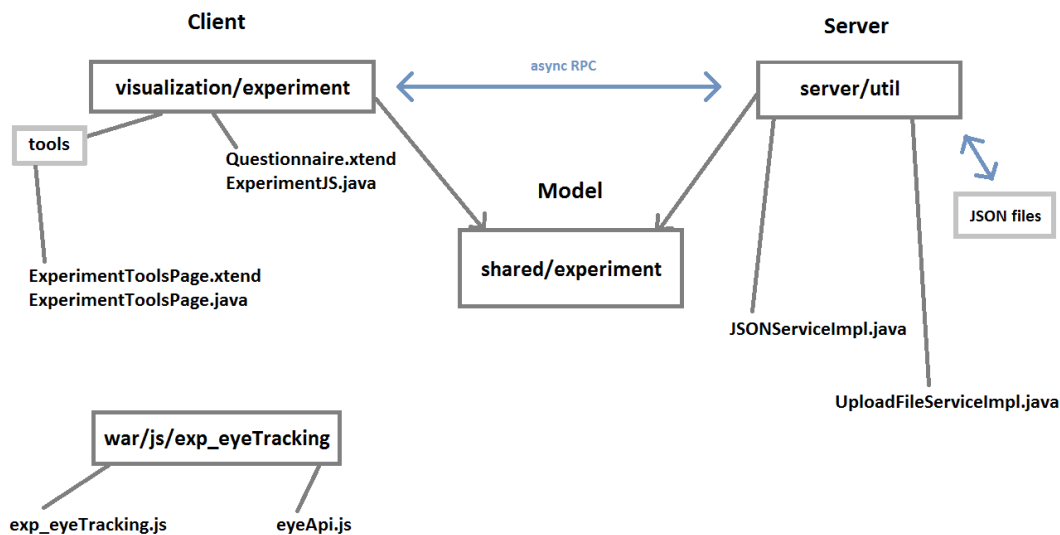


Figure 5.1. ExplorViz Implementation

The Figure 5.1 shows us classes that we will edit for our implementation of the eye tracking extension. In that extension, we want to add not only eye tracking, but also the possibility to record the screen during the participation in an experiment. We also want to make it possible to record the screen when there is no tracking of the eyes. We will modify the experiment overview for the new options of eye tracking and recording of the screen. And we will add these options to the questionnaire as attributes in the JSON files. We are going to implement a new class called `UploadFileServiceImpl.java` on the server side, to

5. Implementation of Eye Tracking Extension

receive and save the big media files of the screen recordings. Like in Chapter 4, there are static files in the war folder for the website. There is nothing to change inside the model side, so we will skip this side in our explanation. We will start with the description of the format of the JSON file, then the server side and what we change there, and it is followed by a description of the client side.

5.1 JSON File

Listing 5.1. Extract of an experiment JSON file

```
1  "questionnaires": [{
2  ...
3      "recordScreen": false,
4      "questions": [{
5          "expApplication": "",
6          "answers": [
7              {
8                  "answerText": "Answer One",
9                  "checkboxChecked": false
10             },
11             {
12                 "answerText": "Answer Two",
13                 "checkboxChecked": false
14             }
15         ],
16         "workingTime": "5",
17         "type": "freeText",
18         "expLandscape": "1467188123864-6247035",
19         "questionText": "Question One"
20     }],
21     "questionnaireTitle": "Questionnaire One",
22     "eyeTracking": false,
23 ...
```

Listing 5.1 shows a section of an example experiment JSON file. We add to a questionnaire the two attributes *recordScreen* and *eyeTracking* as boolean values. One could argue, that if someone wants to track the eyes, they would want to do that in the whole experiment. But we want to give the creator as much freedom as possible. And there exists a possibility that an administrator would want to only track the eye during a specific questionnaire and not in another one.

5.2 Server

Like in the approach in section 3.4 specified with the new questionnaire attribute *preAndPostquestions*, we add get- and set-functions to the *JSONServiceImpl.java* for each attribute, *eyeTracking* and *recordScreen*.

5.2.1 Eye Tracking Data

During the participation of an experiment, as mentioned before, we want to track the eyes and record the screen. These two processes generate data and we want to save the data produced in these processes on the ExplorViz server. In terms of size is the data of the eye tracking small. We can send it to the already existing server class *JSONServiceImpl.java* as string and save it there locally like the JSON data of the experiments. We save it in a folder called *eyeTrackingData* next to where the answers of the participants are saved. This function is called *uploadEyeTrackingData*. As mentioned in the approach, we also want to give an administrator the possibility to look at the data, look at the recording of the screen and at the same time mark where the participant looked at that moment. Therefore we need to offer a get-function for the eye tracking data. This function is called *getEyeTrackingData*.

5.2.2 Screen Recording Data

To get and set the screen recording data is more difficult due to the size of the data than the eye tracking data. To upload the screen recording data, we will create a new class, the *UploadFileServiceImpl.java*. The function takes the data from the request, which is a HTML form element, parses for the file, and saves it locally, like the eye tracking data, in a folder called *screenRecords* next to the answers of the participants. As mentioned before, the size of the data by the screen recording is quite big and in the media format *webm*. More details about the topic of the media format are given later, in section 5.3. So an alternative to transferring the whole file back to the client, is to move a copy to the static files of the website and answer the request with a link to that location. The functionality for this is located inside the *JSONServiceImpl.java*. In summary, we copy the requested file to a specific location of the static files and answer the request with that location. But we do not want to have many copies of the same file on the server, we want to stay as small as possible. Therefore, after copying the file to the specific location, we delete all other files that might exist there. Since the requested file is in use, the server will not delete it.

5.3 Client

On the client side, we need to add the options *eyeTracking* and *screenRecording* to the experiment overview page, to make an administrator able to change them. Then we need to add functions to the current process of displaying an experiment during a participation.

5. Implementation of Eye Tracking Extension

We need to trigger the screen recording and the recording of the eye tracking data. We also need to implement the mentioned external server to communicate with the hardware of the eye tracker, to get the data, mentioned in the approach. And there is also the functionality we want to implement, that an administrator can watch the results of the eye tracking and screen recording together. For this, we need to add an option in the experiment overview page, let the administrator choose between the participates that finished an experiment and then display their recording of the screen and the eye tracking.

5.3.1 ExperimentToolsPage

We want to add the questionnaire attributes *eyeTracking* and *screenRecording* as options to the experiment overview page, as well as the option *Results*, as shown in the mockup in the approach in Figure 3.7. The menu option *Results* should open a modal with the questionnaire participants and their state, if they are done or not with their participation.

Menu Options *eyeTracking* and *screenRecording*

Similar to section 4.4.1, we add HTML menu expressions to the string that gets injected into a modal and then the *RootPanel* of the experiment overview page for both options. And we add for each clickHandler, which opens a modal, informing the administrator that the option was changed and invokes a request to the ExplorViz server to set the toggled value of the attribute. We also reuse the function to toggle and display a tick glyphicon for each option.

Menu Option *Results*

Analog to the section before, we add the menu option to the experiment overview page. There is no need for a tick glyphicon, because this menu option is no attribute that gets toggled. In this case the clickHandler requests a list of the participants. Then we create a HTML table showing the participants, if they are done and if there is eye tracking data of the participant on the server. As shown in Figure 5.2, these columns are indicated with a tick glyphicon in case of this column being true. There are two more columns with their value indicated by a disabled or enabled button. The columns are called *Screen Recording* and declares whether there exists a file of the recording on the server. The last column is a disabled button if the participant did not finish the experiment yet and downloads the experiment and the participants answers and results, the eye tracking data and screen recording, as a zip folder. Noteworthy is here, that we specifically request the server if there are eye tracking data and screen recording saved on the server. In case of the eye tracking data, a false positive is possible. If there is no eye tracking data during the participation, for example there is no eye tracker or the communication fails, the experiment still saves the empty eye tracking data on the server. The data is empty, but the JSON object is there.

5.3. Client

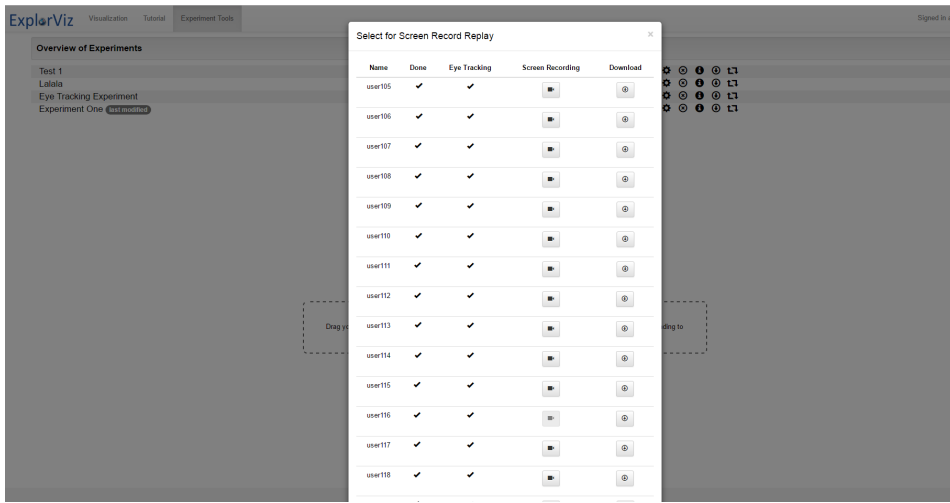


Figure 5.2. Screenshot of a Results Modal

Screen Recording Replay

The button of the column *Screen Recording*, if enabled, opens another modal, a videoplayer with buttons underneath, as shown in Figure 5.3.

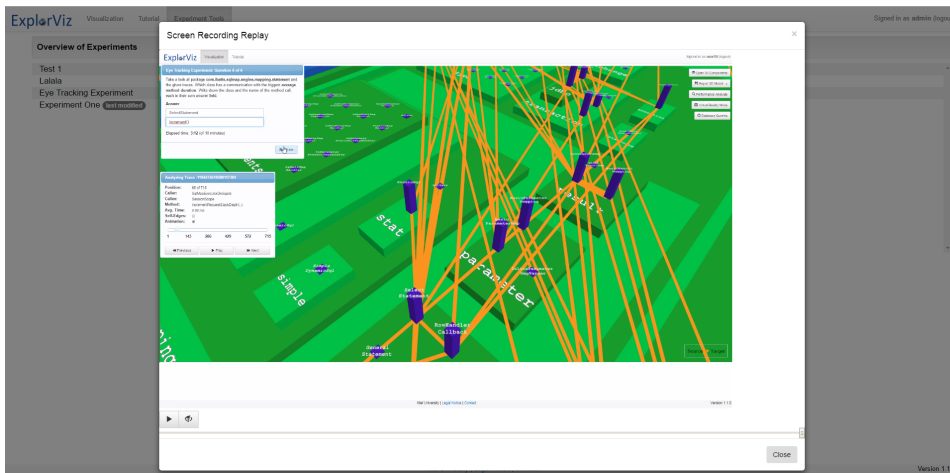


Figure 5.3. Screenshot of a Replay Modal

We control with the buttons both the videoplayer and the replay of the eye tracking data, so we do not show the videoplayer's native controls. We create the videoplayer, a canvas

5. Implementation of Eye Tracking Extension

and the buttons inside a string in the scope of the *ExperimentToolsPage.xtend* and inject it into a modal. Like the *slider.js* in Section 4.4.2 and the Figure 4.3, there exists a JavaScript file outside the scope of ExplorViz. The constructor *startReplayModeJS* gets invoked from inside GWT after the modal is created. His parameters are the eye tracking data, after a request, transferred by the server, and a link to the screen recording media file in the static files of the website. The link gets set as source of the videoplayer.

As shown in Figure 5.3, the button to the left is the start button, showing a play glyphicon, and changes to a pause button with pause glyphicon if clicked. The button on the right controls the display of the eye tracking. Similar to the mockup Figure 3.6, we draw a small translucent red circle on top of the video, onto a canvas. And with the right button the canvas is displayed or hidden.

Listing 5.2. Extract of an eye tracking data file

```
1 {
2   "eyeData": [
3     [
4       0.9116856157779694,
5       0.16776859760284424,
6       1490864805552
7     ],
8     [
9       0.9194962084293365,
10      0.1583147794008255,
11      1490864805567
12     ],
13     [
14      0.9013174474239349,
15      0.1775306835770607,
16      1490864805582
17     ],
18     ...
19     [
20      0.09196797013282776,
21      0.6925255656242371,
22      1490866323934
23     ]
24   ],
25   "width": 1920,
26   "videostart": 1490864814760,
27   "height": 1200
28 }
```

The small bar underneath the buttons controls the progress of the video and eye tracking data. As mentioned before, we draw a red translucent circle with a function called *draw*, which calls itself after a timeout. The draw function also sets the progress bar of the video and manages the synchronization of the drawing of the eye tracking circle with the video time. There are two important functionalities the progress bar has to master. One is the display of the current time of the video and the eye tracking data. And the other is the controlling of the progress. That way a user viewing the replay of a participation of an experiment can fast-forward or go back to another point of the video. In the Listings 5.2, the third number of the list item in *eyeData* shows the *Unix Timestamp*¹ of the computer where a participant took part in an experiment with an eye tracker. We can see, that the eye tracking data entries do not happen in a regular time distance. But we can synchronize it with the timestamps. We know the start point of the recording of the eye tracking data, seen in the Listing 5.2 as attribute *videostart*, and the videoplayer has a time attribute of the duration of the video. The videoplayer also has an attribute *currentTime* that we can access. With these information, we can determine when the eye tracking circle is too far in the future and reset it. After the reset, we go through the whole list until we get to a point where the videotime and the timestamp of the eye tracking data correspond vaguely to another. We make these eye tracking data resets when the timestamp and videotime are more than 40 ms apart from each other.

Listing 5.3. Extract of *exp_eyeTracking* Javascript file

```

1     var videoTime = loadedReplay.videostart + (v.currentTime * 1000);
2     setTimeOfSeekBar();
3     if(currentGaze[2]-videoTime >= 40 && !userOutsideDisplay) {
4         gazeCopy = loadedReplay.eyeData.slice();
5         currentGaze = gazeCopy.shift();
6         console.log("eyeTracking reset " + i);
7     }

```

In Listing 5.3 we see an extract of the file *exp_eyeTracking.js* which is noted in Figure 5.1. The variable *videotime* converts the current time of the video into a representation that we can compare to the eye tracking data timestamp. As mentioned before, we check then whether the video and the eye tracking data, in line 3 of Listing 5.3 represented as variable *currentGaze[]*, are more than 40 ms different from another. Specifically if the eye tracking data is more than 40 ms further than the video. Depending on the gaps between the eye tracking data entries, the reset happens quite often. We see in Listing 5.3, line 3, also a variable called *userOutsideDisplay*. This is a boolean value which indicates whether two eye tracking data entries differ in their timestamp of more than 210 ms. We defined then, that this means, that the experiment participant looked away from the screen and the eye tracker could not get any eye tracking data because of that. The value of 210 ms may

¹<http://www.unixtimestamp.com/>

5. Implementation of Eye Tracking Extension

appear arbitrary, but we checked with example eye tracking data we created during the development and in general transmits the eye tracker its data in intervals under 210 ms. The small bar underneath the buttons controls the progress of the video and eye tracking data.

5.3.2 Experiment

During a participation in an experiment, we need to start the tracking of the eye and the recording of the screen, if the options are enabled. And after the participant finished the experiment, we want to upload the eye tracking data and the screen recording to the ExplorViz server.

Questionnaire

Similar to section 4.4.3, we modify the process during the beginning and at the end of a participation of an experiment. We intervene in the starting process after the attribute *preAndPostquestions* is requested from the server. We request for the attributes *eyeTracking* and *screenRecording* from the server and set them as attributes of class *Questionnaire*.

The starting of the two options must then be done right before the first normal question is injected into the question dialog. In the approach, Section 3.4.3, it was mentioned that the flow of the experiment is interrupted by a window asking for permission to record the screen when the screen recording is started. We want to inject an extra modal before the first question, otherwise would the tracking of the time of the first question already start and pressure the participant unnecessarily when he sees after confirming the security modal, that he had lost time. This would also interfere with the data of the results, since the participant is innocent of the time loss for the first question. As shown in Figure 5.4, the content of the modal announces the begin of the normal questions.

After the confirmation, the first question starts. The screen recording starts after the confirmation of the security modal. An example is shown in Figure 5.5. Before the modal of Figure 5.4 is created and shown, invokes the class *Questionnaire* a native JavaScript function in *ExperimentJS*, which calls similar to Section 4.4.2, a constructor function with one function as parameter, called *EyeTrackScreenRecordExperiment*. The parameter function saves the eye tracking data on the ExplorViz server. Other parameters are the boolean values for *eyeTracking* and *screenRecording*, as well as the Id of the user and a name of the questionnaire.

EyeTrackScreenRecordExperiment

We implement *EyeTrackScreenRecordExperiment* in the external *exp_eyeTracking.js* file. There are different noteworthy aspects that the object takes care of. We start and stop the eye tracking and the screen recording from there, as well as transfer data to the ExplorViz server. And we establish a communication with the before mentioned external communications

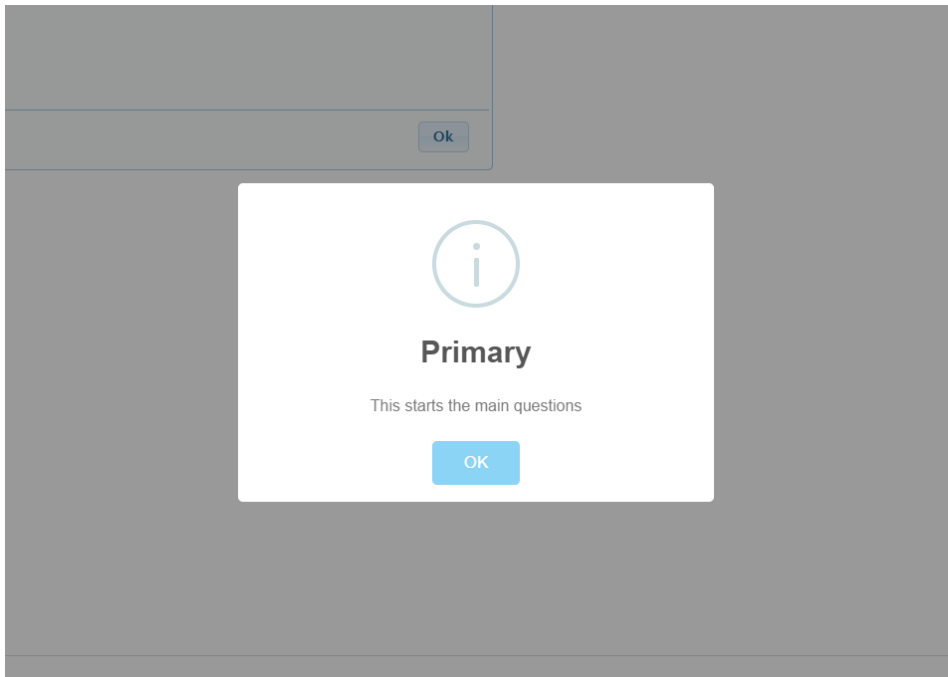


Figure 5.4. Simple Modal before first normal Questions during Experiment

server, which transfers the eye tracking data to us. To stop the recording and tracking, we implement eventListener for the *EyeTrackScreenRecordExperiment* object, since we cannot interact with the object from the GWT side after its construction. As mentioned before, the recorded media files of the screen recording are relatively big. Because of that, we need a process which ensures that the experiment is not finished and closed before the media file is uploaded to the ExplorViz server. We will follow with our description of these mentioned aspects the process during an experiment participation.

It was mentioned before, that we invoke the JavaScript constructor of *EyeTrackScreenRecordExperiment* from the GWT side and receive important parameters, like a function to save the eye tracking data and boolean values for eyeTracking and screenRecording, as well as information of the user and the questionnaire. The screen recording is started first. In Figure 5.5, we see the security modal, mentioned before. As stated before, we use a chrome extension called *Screen Capturing* to get access to the screen. And we use a library called *MediaStreamRecorder.js*, which interacts with the chrome extension and captures the screen for us. We construct the object *mediaRecorder* to use the functions of the library. Noteworthy is here, that we overwrite its function *ondataavailable*, which gets invoked after a set time. When the *ondataavailable* gets called, a part of the screen recording gets send to the ExplorViz server. We transfer parts during the screen recording, because of memory

5. Implementation of Eye Tracking Extension

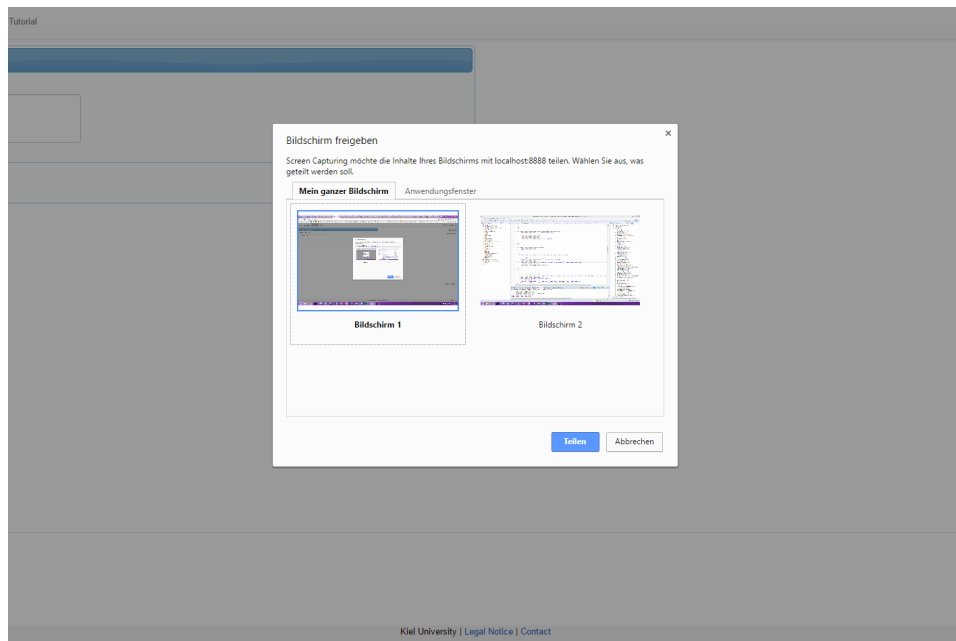


Figure 5.5. Security Modal when starting Screen Recording

restrictions of the browser. We will mention this later again, but if the recording takes too long, the recording object, a so called *blob*, becomes very big and is deleted.

We start the eye tracking by calling the function *startEyetracker*, which we find in the external JavaScript file *eyeApi.js*. This file was implemented by a student in the scope of his bachelorthesis. A connection with a specific local port is established, the before stated external communications server for communicating with the eye tracker. The communication is translated into JavaScript events, to which we let *EyeTrackScreenRecordExperiment* listen. For example, when new eye tracking data is received, did *startEyetracker* implement, that it gets translated to an event called *newGazeData*, on which the document listens and saves the eye tracking data inside the event. Listing 5.4 shows an extract from the *startEyetracker* function, translating received data from the established connection to the external communication server to an event.

Listing 5.4. Extract of eyeApi.js

```

1  connection.onmessage = function (e) {
2      var resp = JSON.parse(e.data);
3      switch(resp.responseType){
4          case "trackerStatus":
5              if(trackerStatus != resp.status){
6                  trackerStatus = resp.status;
7                  $( document ).trigger( "trackerStatusChanged", trackerStatus);
8              }
9              break;
10         case "gazeData":
11             trackerStatus = resp.status;
12             $( document ).trigger( "newGazeData", {"x" : resp.calX, "y" : resp.
13                 calY, "time" : resp.time});
14             break;
15         default:
16             console.log('invalid request type: ' + e.data);
17     }
18 }

```

As stated before, after *EyeTrackScreenRecordExperiment* is constructed, we cannot interact with it via function calls. So we implement a function, which triggers an event called *stopExperiment*. We implement in *EyeTrackScreenRecordExperiment* to let the document listen to this event and react with stopping the screen recording and the eye tracking.

We mentioned before, that we want to implement a mechanism to stop a participant from closing and finishing the experiment before the screen recording is uploaded. Therefore, we create a JavaScript function, which waits for two triggers, before it gives an okay, for the participant to close and finish the experiment. One from the GWT side, after all questions, and if enabled postquestions, were answered. And the other is the upload of the screen recording. We implement this function in *exp_eyeTracking.js* and call it *uploadAndQuestionnaireFinished*. The function lets the document listen to specific events, *uploadFinishedSuccessful*, *uploadFinishedFailure* in case of a failure during the upload of the screen recording, and *questionnaireFinished*. When the participant has finished with the questionnaire, but the upload is not done yet, a simple modal comes up and informs the participant about it. When a failure happens during the upload and the participant ended the experiment, the modal informs him about these circumstances as well. The participant can leave without any modal, if the upload was finished in time.

5. Implementation of Eye Tracking Extension

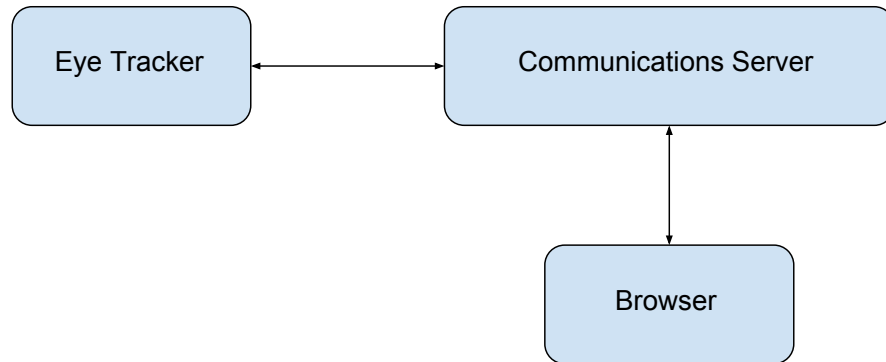


Figure 5.6. Communication on Participants Computer

External Communications Server

As shown in Figure 5.6, the external communications server communicates with the eye tracker via a SDK and with the browser through a specific connection. The source code is in C++ and was implemented by the bachelor student that also implemented *eyeApi.js*. As stated in the foundations, we use a tobii EyeX eye tracker, and include the *TobiiGazeSdk-CApi-4.1.0.806-Win64* as SDK for the tracker. When we started with our development of our approach in November 2016, this SDK was already depreciated by tobii and not available online anymore. Since tobii did not provide any other SDK during that timeframe, we used the SDK the bachelor student used.

The communications server basically establishes a connection to the eye tracker with the SDK, which starts a process that sends data from the eye tracker to the communications server, when the eye tracker notices that it has eye tracking data. The communications server receives this data and transforms it to JSON. Listing 5.5 shows an extract of the transformation process. We can see the attributes of the eye tracking data. After the transformation, the data gets transferred to the browser, due to sending it to the socket and the browser listening to that socket.

Listing 5.5. Extract of eyejsonprotocollserializer.cpp

```

1 void EyeJSONProcollSerializer::serializeTrackingData(QSharedPointer<QPointF>
    gaze, QSharedPointer<QDateTime> timestamp, QSharedPointer<QString> message)
2 {
3
4     QJsonObject json;
5     QJsonValue jResponseType("gazeData");
6     QJsonValue jX(gaze->rx());
7     QJsonValue jY(gaze->ry());
8     QJsonValue jTimestamp(timestamp->currentMsecsSinceEpoch());
9     json["responseType"] = jResponseType;
10    json["calX"] = jX;
11    json["calY"] = jY;
12    json["time"] = jTimestamp;
13    QJsonDocument doc(json);
14    *message = QString(doc.toJson());
15 }

```

Optional Local Improvements

We implemented a process to record the screen and we left the recording in its media format, because the media object inside the browser scope is very big and we had problems with memory restrictions. Our solution was to split the big media file into smaller parts, and implement this idea. We create the parts during the recording and upload them to the ExplorViz server. The problem left is, that during the implementation of the replay of the screen recording, we expected only one media file. For future work, we could implement a mechanism on the server side of ExplorViz to merge the media file parts together. Another problem factor is, that even though we develop and debug on a localhost, the replay of the screen recording still stutters sometimes. This seems to be the case, because the media files are in the *webm* format. For future work, we could implement that the media files not only get merged for one user, but also gets transformed in the more common *mp4* format. But it is also possible to install software and do it on the local computer.

Evaluation

In this chapter we evaluate our implementation. We conduct an experiment for the sake of testing the functionality of our implementation. We describe its design, its results and discuss them. The goal of our evaluation is, to test the functionality of our implementation and assess the complexity for analyzing the eye tracking data and screen recordings.

6.1 Experiment Design

An experiment needs to be developed first and our design decisions have to be documented. Then we will describe and show our tasks, describe independent and dependent variables influencing the experiment, the treatment during the experiment, and the participants.

6.1.1 Research Questions and Hypotheses

The experiment is based on three research questions, which we try to answer by conducting the experiment.

1. What is the ratio between participants gazing not at the display, gazing at the question modal, and the ExplorViz interactive interface?
2. Is there a correlation between correctness of answers and amount of gaze time on the question modal versus on the interactive interface?
3. What does a participant mostly do (gaze and interaction with interface) when answering a question incorrect?

Further, we define two hypotheses:

1. The ratio of gazing at the question modal, interface and not at the display differs between participants.
2. The correctness of answers differs between participants that gaze more at the question modal than the interface and vice versa.

And we specify accordingly two null hypotheses:

6. Evaluation

1. The ratio of gazing at the question modal, interface and not at the display does not differ between participants.
2. The correctness of answers does not differ between participants that gaze more at the question modal than the interface and vice versa.

6.1.2 Empirical Methods

The empirical method we use for trying to answer our research questions and proving our hypotheses, we will use an experiment. Not only do experiments help us to evaluate, according to [Basili et al. 1986], through gathering data but we can test the functionality of our implementation. We will use tasks, that cover different features of the ExplorViz interactive interface. And participants can vote afterwards the difficulty of the tasks. The eyes are tracked during the questionnaire, as well as the screen.

6.1.3 Tasks

Our goal for the experiment is to find out whether our hypotheses are correct and to answer our research questions, while taking into account that we want to test the functionality of our implementations. To our knowledge there does not exist an experiment which checks whether eye tracking is valuable for a non static interface. We develop the tasks with the goal, that the participants have to use various methods and features of the ExplorViz tool. The displayed software landscape is a monitored JPetStore instance, that we use because it makes a slightly complex software landscape inside ExplorViz. The tasks are in a sequence to build up in difficulty. They start with easy tasks. All tasks require at least two answers, which are displayed as text input fields. The amount of answers the participants need to give is displayed in the amount of text fields shown. They act like a hint for the correct answers. Furthermore, we choose text input fields to make it impossible for participants to guess a correct answer when there are given answers from which they just have to choose.

We want to introduce the participant with the first question. The participant has to open some packages and gains an overview of the packages and classes. Through the second and third tasks, gets the participants even more information about JPetStore, and lets the participant search through the displayed components. For solving task T2, must the participant understand, that not all communications with other class are shown when parent packages are closed. The communications are summarized to one line then. And this is important to understand, when searching for classes to improve. Task T3 concerns following the communications line and using a specific tool of ExplorViz, the trace analyzer. But it can also be correctly answered if the trace analyzer feature is not used and only following a shortest possible path. The last task makes use of the communication summary when a parent package is closed. But through clicking each communication in the context, this solving is also possible and the participant needs find the correct column of an information modal.

6.1. Experiment Design

As shown in Table 6.1, we assigned to each task an id, that we will use to refer to the tasks. And we have assigned a score for each task. They are the points we give if the task was solved fully correct. Our evaluation schema for the scores is the following. For tasks T1.1 and T1.3 we grade one point only if the answers are correct. In task T1.2 it is possible to get one point, when one answer is correct and two points if both are correct. The schema for task T2, we grade one point for two correct answers of five. Furthermore, we grade two points for three correct answers and three points if all five answers were correct. Task T3 is graded only full two points if the path is correct and we grade one point for task T4 if one of the answers is correct and two points if both answers are correct.

Table 6.1. Description of the Task for the Experiment

ID	Description	Score
	<i>Context: Gaining an overview of interface</i>	
T1.1	Take a look at the packages. There are two packages named 'impl'. Write their hierarchy in form 'JPetStore.com.ibatis.xx' down.	1
T1.2	Take a look at package com.ibatis.jpjpetstore . Which classes have the lowest and highest amount of active instances. Ignore the classes with 0 active instances. Write only the class name and start with the class with the lowest.	2
T1.3	Take a look at the given data of the whole application. Between which classes exists the most communication (requests)?	1
	<i>Context: Identifying refactor potential</i>	
T2	Take a look inside package com.ibatis.sqlmap.engine.mapping . Which classes have high fan-in (at least 3 incoming communications) and almost no fan-out (at max. 1 outgoing communication) with other classes ?	3
	<i>Context: Following traces or communications</i>	
T3	Take a look at class 'CatalogBean' in package com.ibatis.jpjpetstore.presentation and class 'DaoImpl' inside com.ibatis.dao.engine.impl . Name the classes which are between a communication of 'CatalogBean' and 'DaoImpl'. Start with naming 'CatalogBean' and write down the shortest possible path. (Hint: you can use the traceanalyser.)	2
	<i>Context: Determining details</i>	
T4	Take a look at package com.ibatis.sqlmap.engine.mapping.statement and the given traces. Which class has a communication with the biggest average method duration . Write down the class and the name of the method call, each in their own answer field.	2

6. Evaluation

Dependent and Independent Variables

The dependent variables in the experiment are the correctness of the answers and the time spent on each task. And another is where the participant gazes on the screen during the experiment. The independent variable is the tool ExplorViz that the participants use for solving the tasks.

Treatment

All participants used ExplorViz to solve the program comprehension tasks after they finished an integrated interactive tutorial. Before they started with the tutorial, they completed a calibration of the eye tracker.

Personal Information

Before and after the tasks, we will ask the participants personal questions. The initial questions demand statistical information, like gender, age, and highest completed degree. Also we request them to perform a self-assessment in regard to their experience with the monitored application, JPetStore, and with the ExplorViz tool. The scale that they could choose from for their experience of JPetStore ranged from 'None', 'Beginner (saw source code)', 'Regular (implemented a feature)', 'Advanced (implemented more than one feature)', to 'Expert (several years of experience)'. The options for the assessment of their experience with ExplorViz ranges from 'None (never used it)', 'Beginner (used it casually)', 'Regular (familiar with some features)', 'Advanced (familiar with most features)', to 'Expert (years of experience)'. We referred here to the 5-point approach of [Likert 1932].

After the questionnaire, the participants are questioned about their experiences during solving of the tasks. They are asked about the pressure they perceived from the given time, the tracking of the eye and the recording of the screen. And afterwards about their view of the difficulty of the solved tasks. They could choose for all answers concerning the difficulty of the tasks, on a scale from 1 (too easy) to 4 (too difficult), using a 4-point Likert Scale [Likert 1932]. In case of the perceived pressure, the scale went from 1 (no pressure), to 4 (high pressure) and is also a 4-point Likert Scale [Likert 1932].

Population

We asked students, PHD students and research assistants for their participation. We informed the future participants, that we will provide them during the experiment with cookies and chocolate bonbons. 20 persons participated.

Due to bugs and an accident where the eye tracker was moved by a participant, five participants had to be eliminated because we did not have any screen recording or usable eye tracking data as results for the participants.

6.2 Operation

The following sections explain how we executed our experiments.

6.2.1 Experimental Set-up

In this part, we describe relevant hardware and software we used for the experiment. The computer we used, had a 24 inch monitor with a resolution of 1920×1200 pixels. The operation system is a Microsoft Windows 10 64-bit version with an installed JDK(Java Development Kit) in version 1.8. Further, as mentioned before, we installed an eye tracker, the tobii *EyeX Controller*. And we installed the chrome browser, where we installed the chrome extension *Screen Capturing*. And we need to start the communications server locally on the computer. Further information on install instructions and versions of the installed soft- and hardware are described in Appendix A.

6.2.2 Create Input

We created the execution trace based on a running instance of the JPetStore 6. The JPetStore was instrumented to be monitored with ExplorViz. During the execution of the JPetStore instance, the monitoring is done and we interacted with the JPetStore web application, producing process chains, producing the execution trace that we used.

6.2.3 Tutorial

We introduced the participants to the tool ExplorViz with the guided and interactive tutorial ExplorViz provides. The tutorial showed the functionality that the participants would need to solve the tasks.

6.2.4 Questionnaire

We used the electronic questionnaire integrated in ExplorViz, which we extended in this thesis. That way we have the answers in digital form, track the time and track the participants eyes during their reading and answering of their tasks and the participant is forced to enter answers into the given number of answer input fields.

6.2.5 Pilot Study

Before we performed our experiment, we asked two people to perform as participants in a pilot study. We checked whether the assigned time to each task is reasonable, if the tasks and tutorial texts are comprehensible and if there are any careless mistakes left in the tutorial or tasks. We changed the tasks and tutorial according to their feedback and added a task regarding the structure of packages in the software application view, see task

6. Evaluation

T1.1, and added hints to other tasks. Task T1.1 is added, to introduce the participants into the software structure. The hints, one explicit and other implicit through bold letters, are added to mark important parts of the question. It was indicated that they are otherwise easily overlooked.

6.2.6 Procedure

The experiment took place at CAU Kiel with one computer. One participant could take part in the experiment at a time. The participants were provided with cookies and chocolate bonbons during their participation. They started with reading information about the experiment, which is located in Appendix B. It describes that they can cancel their participation anytime and that we use their data anonymously. Also why we are carrying out an experiment and that their eyes are going to be tracked and the screen is going to be recorded. We told them to tell us, when the security modal for the screen recording appears, so we as the attendants are sure they clicked the right button. Then we start the calibration of the eye tracker for the participant. The tobii EyeX Tracker we use provides a calibration process and we use it for the participant. Afterwards we put the browser in full-display mode (F11) before the participants log in and they start the tutorial. The experiment's questionnaire starts afterwards, first the prequestions, then the tasks from Table 6.1 and the postquestions afterwards. The first task from Table 6.1 appears, shows up a security modal asking permission to transmit the content of the screen to the ExplorViz server, which the participant confirms under our supervision.

We assigned the first task 8 and the Second 7 minutes. For all other tasks are 10 minutes designated.

6.3 Data Collection

We collected during the experiment the answers of the prequestions, the normal questions and the postquestions. We also tracked the time participants needed for each task and tracked their eye during solving of the normal questions. Additionally we recorded the screen during the time the participant's eyes were tracked.

6.3.1 Timing and Tracking Information

The tracking of the time that each participant takes, is done automatically by the electronic questionnaire system of ExplorViz. The recording of the screen is done with the help of a chrome extension, see Section 2.8, to see what the participants did during the experiment. And we track the eyes of the participant with an eye tracker, which we use in connection with the recorded screen to see where participants looked during the experiment. We will use this data to discuss our hypotheses later in the section Discussion.

6.3.2 Correctness Information

Each task has a specific amount of correct answers, which were displayed in the questionnaire with empty text input fields. The participants usually understood, that they needed to fill all input fields with answers. The answers are often restricted with example formats given in the question.

6.4 Results

In this section we describe the results we have from our experiment. After some bugs, which we will mention later, and an oversight where the eye tracker was accidentally moved, we had to eliminate five participants, because not all data from them was available. Thus, we have results of 15 participants. We will start with presenting the prequestions, then show the results of the tasks and afterwards of the postquestions.

Prequestions

The questionnaire starts with the prequestions, and asks the participants to assess their experience with the application JPetStore and the tool ExplorViz from 'None' to 'Expert'. We mapped in an approach for results the numbers 1 to 5 to the assessments, with 1 representing 'None' and 5 'Expert'. The mean for the experience with JPetStore is 1,2667, and the mean for ExplorViz is 1,8667. Four participants indicated that they have seen source code of JPetStore, the others had no experience with it. In comparison, the majority of the participants had no experience with the ExplorViz tool. Less than half of all participants have used it casually or were familiar with some features. Two rated themselves as advanced users, feeling they are familiar with most features of the ExplorViz tool.

Tasks Correctness

Table 6.2. Correctness Results of Participants

ID	Mean	SD	Score
T1.1	0,8	0,4332	1
T1.2	1,2667	0,9501	2
T1.3	0,5333	0,5001	1
T2	1,6	0,8664	3
T3	1,8667	0,662	2
T4	1,0667	0,9356	2

In Table 6.2 we see the results concerning the correctness of the tasks, a mean and a standard deviation for each task. We graded the answers of the tasks according to the

6. Evaluation

points we assigned the tasks. Noteworthy is here, that we gave in T3 task 4, 2 points for 3 correct answers of 5 and 1 point for 2 correct answers. 3 points are given only if 5 of 5 answers in T3 are correct, this happened only once. And in T4, task 5, we assigned only 2 points or 0. We assigned 2 points to the task, because of the difficulty and the answers is a path of one correct answer, so we marked it either correct with full points or incorrect with 0 points. All individual results are documented in the Appendix C.

Eye Data Results

For analyzing the data, we developed a website with local JavaScript inside to compute our results. It is located in the git repository¹. We enter the data, the offset at the beginning of the video and for each time the participant moved the question modal, a start and end point in format $\langle \text{minutes}, \text{seconds} \rangle$ and the upper left and down right coordinates of the new location of the question modal in format $\langle x, y \rangle$. The x and y numbers are assessed with another website (*testCanvas.html*) and opening the respective media file of the screen recording to compare the locations of the question modal with a displayed line. The results of all participants concerning the ratio where they looked is in Appendix C.

Table 6.3. Information Ratio of Eye Tracking Data

Category	Mean	SD
Question Modal	17,0679	8,0997
Interface	78,0356	20,0754
Non-Display	4,8965	4,137

The Table 6.3 shows in the first line the mean of the percent that the participants looked at the question modal, during the solving of the tasks. The define the mean as the average of all participants. And the results in Table 6.3 are measured in regard to percentage. We computed the percentage value for each category, for each participant and then calculated the mean and standard deviation. The category *question modal* represents, the amount of time, in percentage, the participants looked at the question modal. Category *Interface* represents the amount of time that participants looked at anything else on the display, thus the interface of the ExplorViz tool. And *Non-Display* shows the amount of time, the eye tracker could not track the eyes of the participant and thus means, they looked away from the computer monitor, for example to the keyboard. We defined, that whenever the eye tracker could not track the eyes for longer than 210ms, the participant looked not at the display.

It is noteworthy, that the standard deviation of the Interface in very high.

Table 6.4. Postquestions Mean and Standard Deviation

Category	Mean	SD
Pressure Time	1,733	0,7988
Pressure Eye Tracking	1,2	0,6399
Pressure Screen Recording	1,467	0,7988
Difficulty T1.1	1,933	0,4577
Difficulty T1.2	1,933	0,4577
Difficulty T1.3	2,333	0,8997
Difficulty T2	2,929	0,8287
Difficulty T3	2,429	0,9376
Difficulty T4	2,857	0,663

Postquestions

Table 1 shows the mean and the standard deviation of the assessment that all participants gave for perceived pressures and difficulty of each task. The pressures are scaled from 1, for no pressure, to 4, for high pressure. The pressure for time is nearly *a little* for the mean, and the pressure for eye tracking is with a mean of 1,2 the most near to the assessment choice of *no pressure*. The screen recording is in comparison slightly higher with a mean of 1,467, resulting in the assessment choice of *a little* if round up.

The lowest mean have tasks T1.1 and T1.2 with 1,933, which is nearest to assessment choice *appropriate*. The highest mean with 2,929 is task T2, which is nearest assessment choice *a little difficult*.

6.5 Discussion

In this section, we discuss our results that we presented before. We are interested whether there is a correlation between correctness of a participant and their amount of time looking at the question modal. In case of the results of the prequestions, the information about the participants experience, would appear to correlate with the correctness concerning their experience with ExplorViz, if it is higher than a 'Beginner'(1). The correctness is clearly higher than average in case of our four participants where this applies.

6.5.1 Correctness of Tasks

Figure 6.1 shows, that the correctness decreases during the questionnaire. This implies, that the first tasks were appropriately set to the beginning of the questionnaire. The first three tasks were created to get a rough overview of the software. Many participants could not

¹[xxx/AnalyzeExperiment/getRatio.html](#)

6. Evaluation

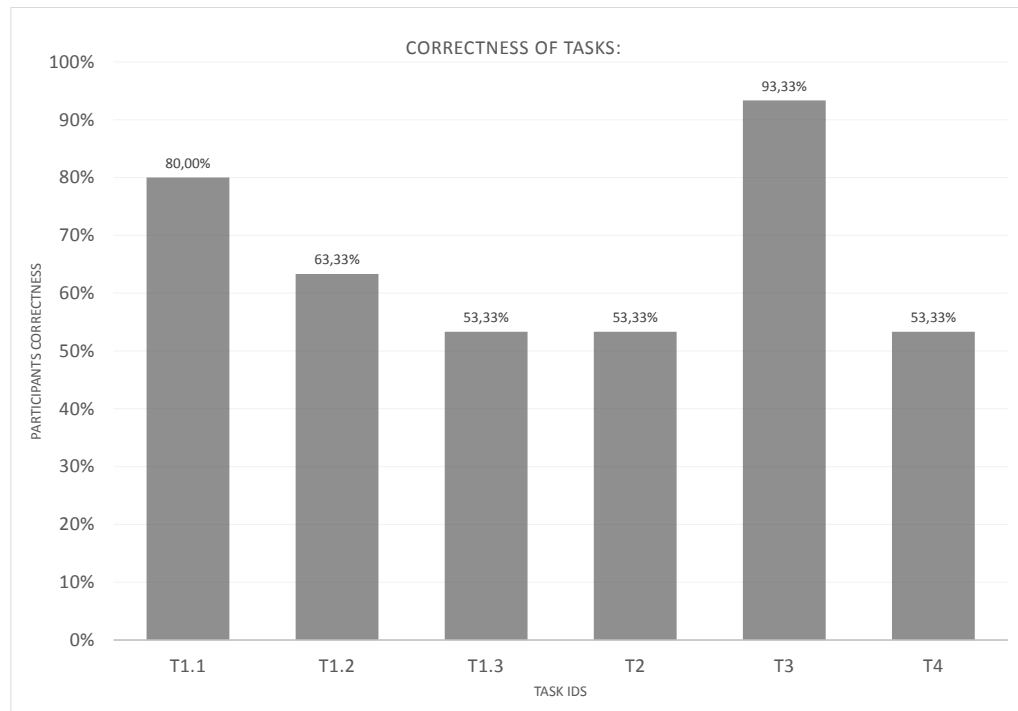


Figure 6.1. Correctness of Tasks

name all answers for the task T2, because they did not open all linked packages, as hinted in the question. The hint is, that the word *classes* was in bold letters, see table 6.1. This might have been too difficult to detect, since the majority, as seen with the prequestions, are novices with the ExplorViz tool. In comparison, the task T3 achieved the highest correctness in our experiment with an average correctness of 93,33% and the following task, T4, has very little correctness with 53,33%. With the exception of task T3, the correctness of the tasks implies a correlation with the participants' perceived difficulty.

6.5.2 Eye Tracking Data

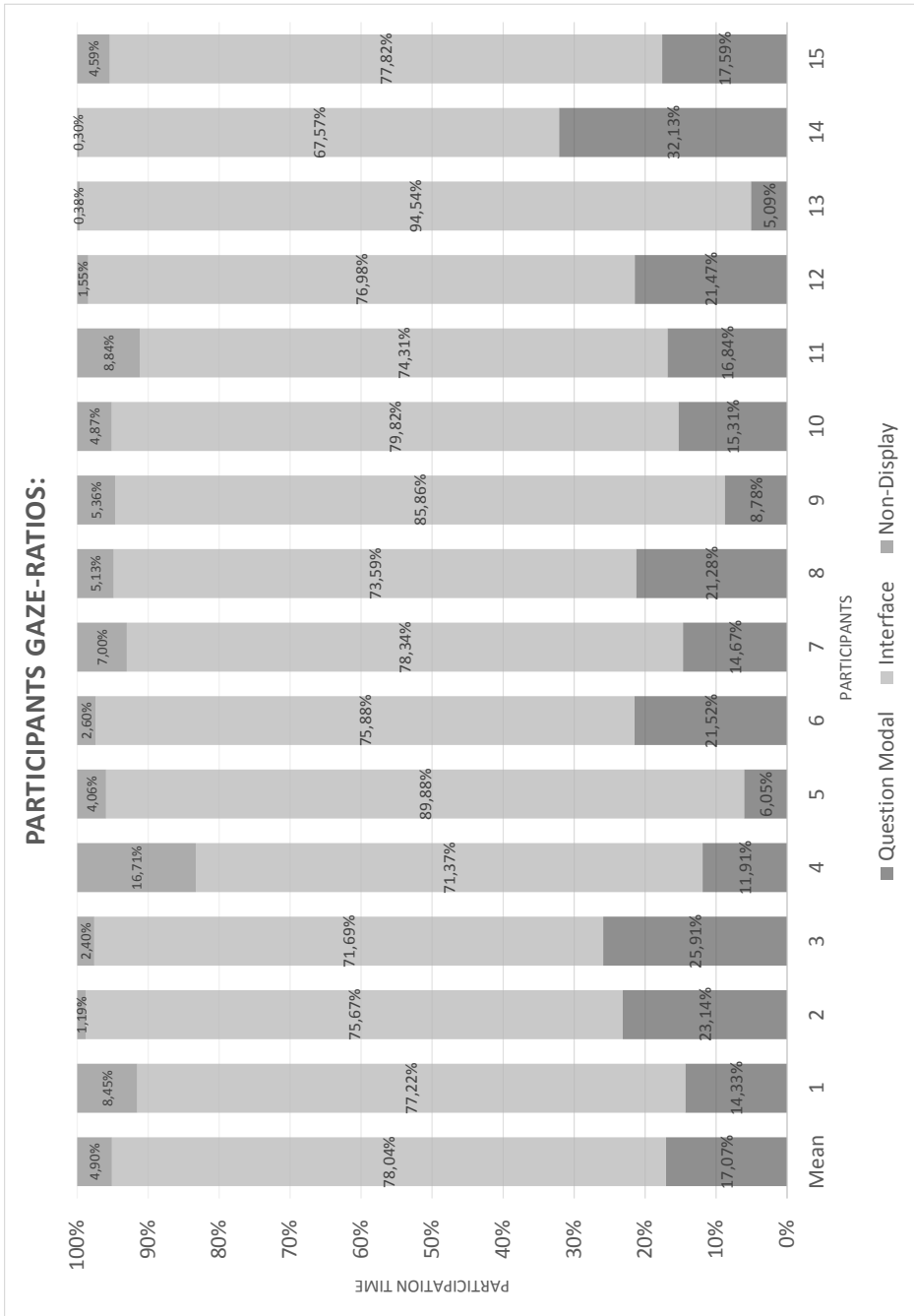


Figure 6.2. Information on the Participants Gazes

6. Evaluation

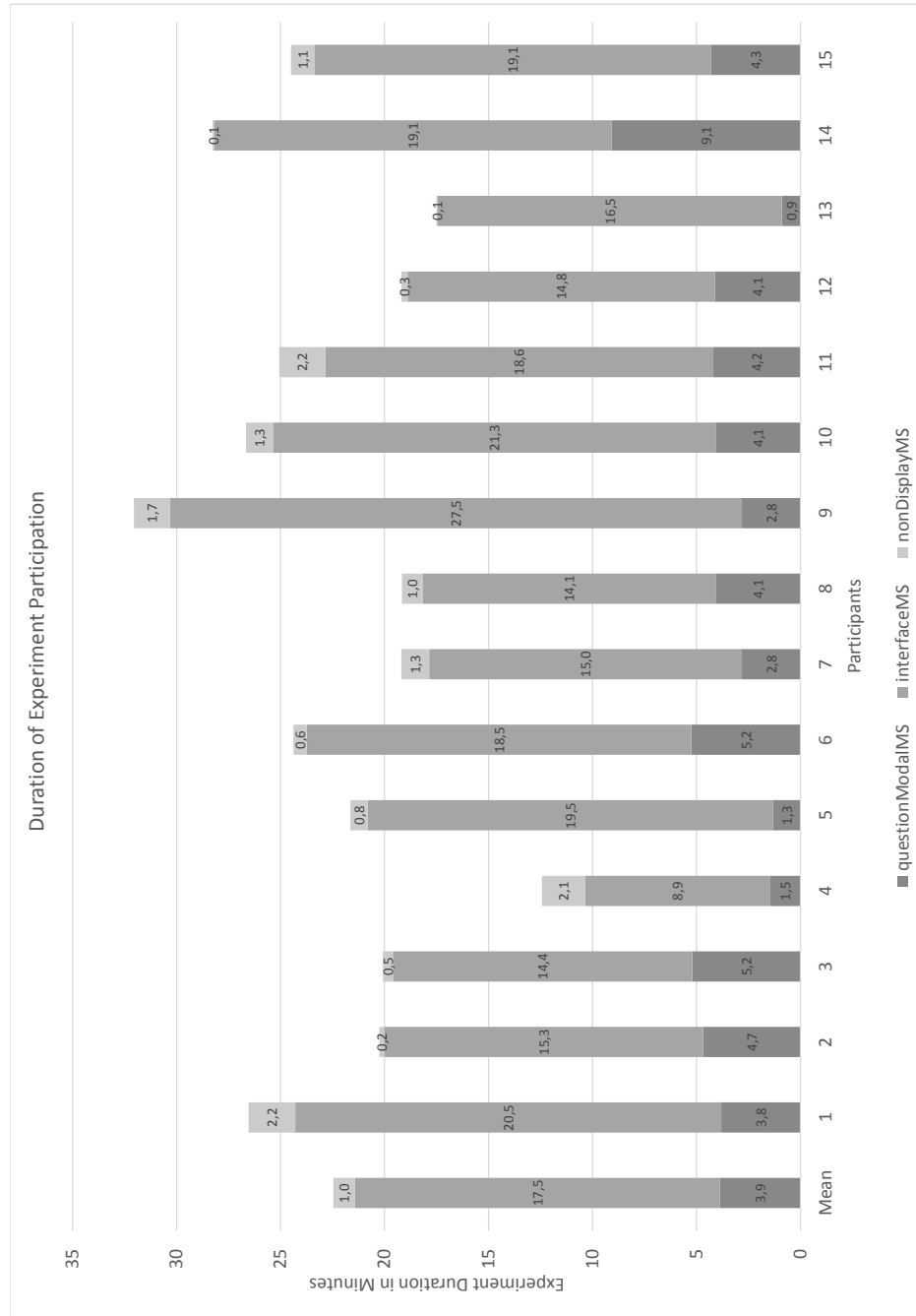


Figure 6.3. Duration of Experiment Participation

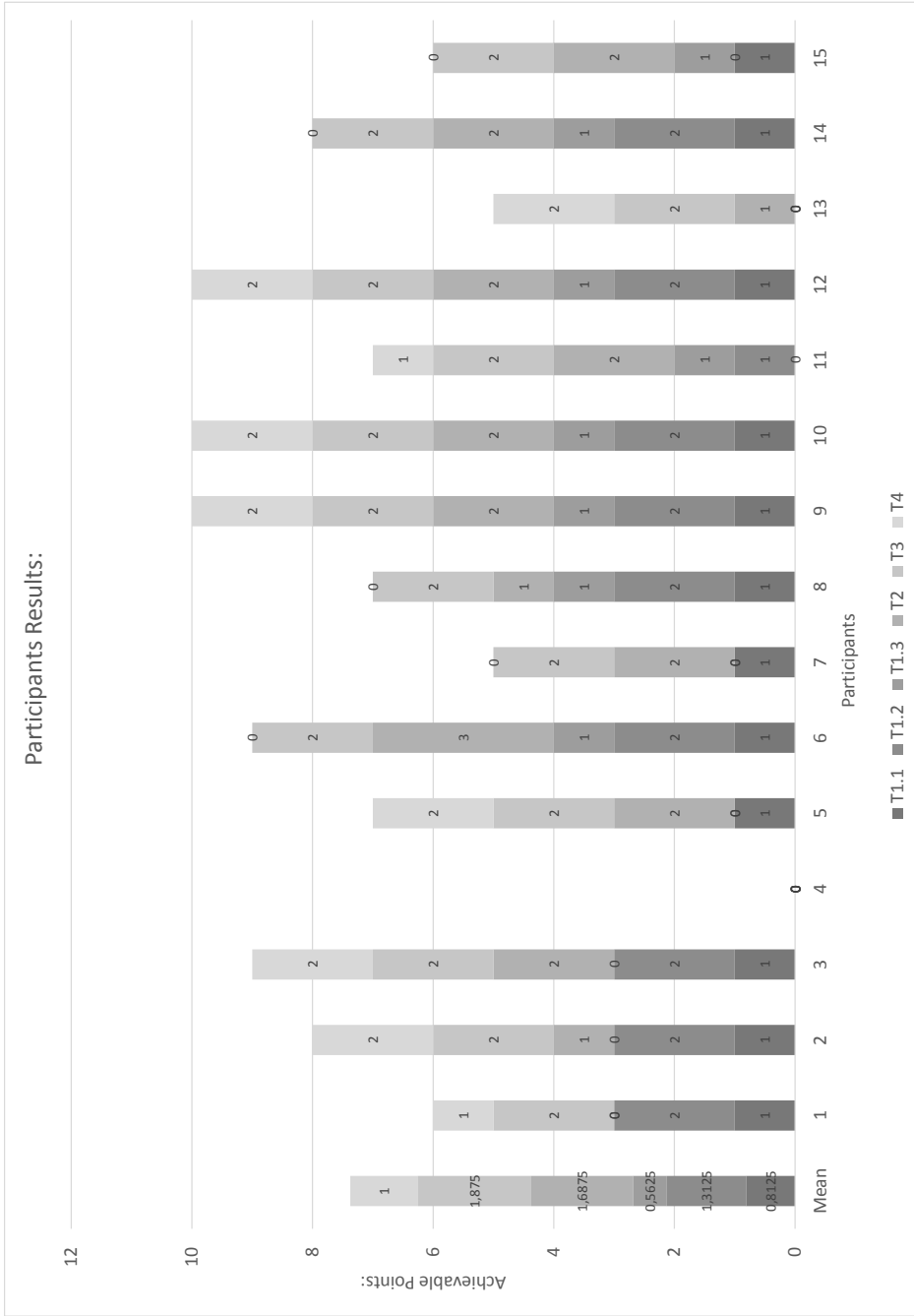


Figure 6.4. Participants Individual Correctness Information

6. Evaluation

As shown in Figure 6.2, the amount of time participants spent on looking at the interface, the question modal, and not at the display, in relation to their respective time they took the experiment, differs from each other. Figure 6.3 contains shows also the for the participants the amount of time they looked at the question modal, the interface and not at the display. Figure 6.3 displays the variables in minutes.

In Table 6.3 we can see that the standard deviation is especially high for the amount of time looked at the *interface*. In the beginning of this chapter, we constructed the hypothesis, that participants have different ratios concerning question modal, interface and non-display. Our data concerning the ratio and the standard deviation imply, that they do differ, in case of our small experiment.

In Figure 6.4 we see the participants individual points they got for each task. This represents their correctness individually. We can see in Figure 6.4, that there is one participant that got 0 points. Participant number 4 has a relatively high amount of time, in which she did not look at the display. This might be a correlation, but we do not have enough data to make this assumption. Another participant that has a small correctness, is participant number 13 with 5 points. The amount of time spent looking at the question modal is in comparison to the mean of everyone, 17,07%, small with 5,09%. In comparison, participant number 14 looked relatively often with 32,13% at the question modal. With his eight point, it is slightly above average and one of the seventh best in regards to correctness. The other participant that has the same amount of points, is participant number 2. We developed a hypothesis, which implied that the gaze ratios of participants with the same correctness are similar. The ratio of looking at the question modal is for participants number 2 and number 14 both higher than average, but they differ 8,99% in their amount of time looked at the question modal. This implies that that there might be no correlation between points and amount of time spent looking at the question modal.

6.5.3 Postquestions

In Figure 6.5 are the results transformed into percentages, showing of, how the participants assessed the difficulty of each task. This assessment was filled in after they solved the tasks. The percentage ranges from 0% as easy to 100% as very difficult. As shown in Figure 6.5, are the tasks rated in general as not easy and mostly not too difficult. The difficulty increases from the first to the last task, except for task T2, where the assessed difficulty is the highest. In Figure 6.1 is shown, that the correctness was relatively small in task T2, therefore it seems plausible to say that the task was difficult.

Task T3 correctness is according to Figure 6.1 with 93,33% even higher then the first task T1.1 with 80%, which was supposed to be the easiest task. With a perceived difficulty, third highest with 47,62% in Figure 6.5, the task T3 seems to be a difficult task which on average still got solved correctly.

6.6. Threads to Validity

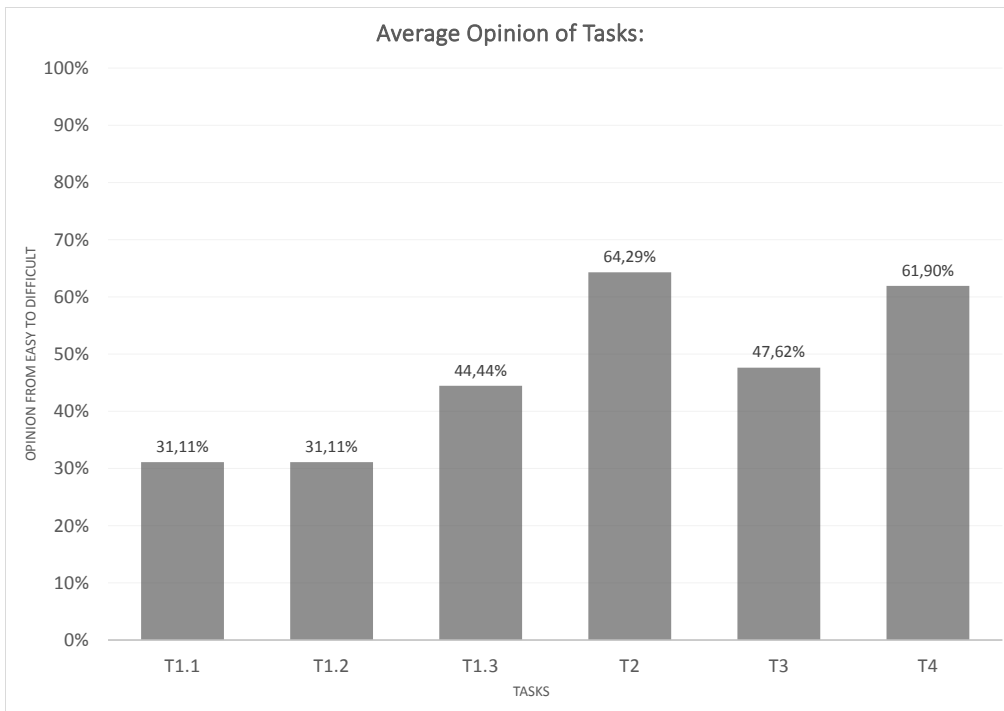


Figure 6.5. Assessed Average Difficulty of Tasks

6.6 Threads to Validity

This section addresses experiment conditions that might threaten the validity of or influenced our results of the experiment, (internal and external validity) [Shadish et al. 2002].

6.6.1 Internal Validity

Reasons threatening the internal validity occur from the experiment and are explained in the following sections.

Participants

Our participants were not many and they had various degrees of expertise. Our results and analysis could be different with more participants or when the participants had more expertise with the ExplorViz interface or software in general. Another threat is their

6. Evaluation

motivation. We provided cookies and chocolate, and some were interested in the subject, but this experiment provides no value to their professional life. The results could differ if the participant's motivation was higher.

The participants rated the average pressure they perceived in regard to the time for each task as less than *a little* in average. The ratings for the eye tracking is *no pressure*, except for three participants, who experience *a little* pressure. And the pressure concerning the recording of the screen is in average *no pressure*, but some did feel *a little* pressure and one experienced more pressure and rated it as *maintainable*. Therefore, if more or less pressure would have been experienced, the results could have differed.

Another reason that might have influenced the participants, is that most of them have a relationship with the attendant and creator of the experiment. This might influence the participants to rate the tasks difficulty more as *appropriate* than difficult. If they did not know the creator and attendant of the experiment, they might have rated more honestly.

A thread that concerning the participants is also, that their head movement and maybe gestures with their hands might have influenced the results. Because of their movement or hand gestures, the eye tracking data could be different and if they might not have moved, the results might have been slightly different.

Tasks

A threat to our validity is that the tasks might have been too difficult. The assessment and correctness of the tasks, Table 1 and Table 6.2 respectively, do show that task T2 and T4 were perceived as difficult and the incorrect answers imply this as well.

Tutorial

Another reason why participants might answer differently, is if the tutorial had a higher quality. We noticed during our experiment, that the tutorial did explain all necessary features and methods to solve the tasks, but that ExplorViz allows users to use several shortcuts to get to the answers more efficiently. For one, the possibility to right click and turn the perspective inside the application view was not mentioned in the tutorial. As stated, the tasks were still possible to solve, but using the feature to turn the perspective might have made solving them easier and even faster. And another thing is, that there exists a button in the interface to open all packages, on every layer, at once in the application view. That way we can view all classes at once and do not have to open every one of them, one after the other. If this was mentioned in the tutorial, the least correct answered task T2 might have been solved successfully more often. As stated, it was possible to solve the task, but without the button more classes had to be opened manually.

Some participants mentioned afterwards, that even though it was shown in the tutorial, they forgot during the solving of the tasks how to open the trace analyzer feature. If the tutorial would take that into account and improve the way it explains this feature, this might positively influence the results.

6.7. Lessons Learned and Challenges Occurred

Miscellaneous

An influence on our results could be the different kinds of pressures the participants experienced during the experiment. They could have felt pressure in regard to the time they had for each task, the participants might have assessed the tasks differently because the time limit was set too small or too big. And also, if the eye tracker were more precise, our results might have differed concerning the looking ratio results. Another thread is that our analyzing script for computing the results of the eye tracking data could be incorrect.

Some participants dragged the question modal to another place during the experiment. We mentioned in Section 6.4 that we followed the question modal's movement to compute the correct ratio. We tried our best, but if some software would have tracked the absolute correct coordinates, also during the movement, the results might have been more precise.

6.6.2 External Validity

According to [Shadish et al. 2002] experiment results can not always be generalized. For one, the tasks we developed are dependent on the software landscape. The structure and size changes between different systems and JPetStore, on which basis we created our traces, is a sample software. So our results may not be applicable for other software systems.

Another reason why our results may not be applicable to generalization, is that the participants in the majority did not know the ExplorViz interface beforehand. They used it for the first time and we do not know if the results might be influenced if they would use the ExplorViz interface more often and in a professional sense. To eliminate this assumption we would need more experienced participants in another experiment.

Participants are situated in unusual circumstances during the experiment. In a way, the employed tasks might not reflect real program comprehension tasks. They animate the participants to use the features of the ExplorViz interface but might not be applicable to real tasks that professionals would operate. So without further experiments with professionals and validated tasks, our results might not be applicable to generalization.

6.7 Lessons Learned and Challenges Occurred

We experienced several challenges during the experiment. At first, some spelling mistakes were not noticed during the pilot study inside the tutorial. They were not grave but inconvenient. Also one participant moved the eye tracker after their calibration and because the website is always in full screen, it was not noticed that the eye tracker could only rarely track the eyes of the participant, resulting in unusable eye tracking data. Four participants experienced a bug that was undiscovered within our implementation. When a participant takes longer than expected, the communication between the ExplorViz client and the related server happened to be disconnected and a failure occurs during the upload of the screen recording. The problem handling should intercept there and download the screen

6. Evaluation

recording locally to the computer, but this command was implemented in a wrong scope, so the screen recording is empty and is lost. This was repaired during the experiment.

Two out of three participants experienced a specific bug during their interaction with the ExplorViz tool. This bug is replicable, if one uses the trace analyzer, the interface zooms into the software application view and follows the trace that one chose. If we try to zoom in or out for a time and then close the trace analyzer, the interface does not react anymore to left or right click. This can be solved with a workaround, to switch to the landscape view and then back into the application view. This bug lead to some confusion for the participants.

Analyzing the eye tracking data was interesting but complex. There is no alternative to the numbers we compute, so we do not know if there might be a miscalculation and our results concerning the ratios might be incorrect. Also, identifying the coordinates for the question modal as well as the points in time that they were moved, was time-consuming and vague. This can be improved if the question modal could not be moved by the participants.

6.8 Summary

In summary, we have lots of data implying that the tasks T2 and T4 might have been too difficult. The difficulty of the first three tasks are perceived to increase in order and task T3 was perceived as difficult but the correctness was in comparison to the other tasks, the highest. Thus we could task T1.1, T1.2, T1.3 and T3 as appropriate.

We found data implying one hypothesis of our experiment design, that the amount of time looking at the question modal, the interface and not at the display, do differ from each other from one participant to another. And the hypothesis that there exists a correlation between the ratio of looking at the question modal and the correctness of the tasks, is implied by the data to not be true.

With our experiment and its result we confirmed the functionality of a part of our implementation. This part is the display during an experiment. During our evaluation and analysis of the results of the experiment, we confirmed the functionality of the other part, acting as an administrator.

Related Work

There are several works with experiments in ExplorViz, like the master's thesis [Finke 2014]. She presented an approach to conduct interactive integrated experiments in ExplorViz and evaluated her approach with controlled experiments. In her experiments, she compared ExplorViz with another visualization tool called EXTRAVIS. Our approach bases on her work, and we do not compare ExplorViz with other tools in our thesis. Our goal is, in comparison, to determine whether the feature of eye tracking and screen recording enhances the experiment mode of ExplorViz.

In the work of [Fittkau et al. 2013], there were two controlled experiments conducted, also with screen recording. The screen recording data was not used and existed to be looked at when a participant answer is unusual. In comparison, we explicitly used the screen recording for our analysis of the eye tracking data.

There are not many works which engage in eye tracking experiments with interactive interfaces. One is the work [Kevic et al. 2015], where they conducted an exploratory study to determine what developers explicitly do during a change task. They used very fine-granular eye tracking data and they are automatically linked to the respective object, that they looked at in the source code of the IDE. One of their findings is, that the eye tracking data contains other facets than interaction data. In comparison, our experiment was conducted with the subgoal to find correlations between the amount of time a participant looked at the question modal, interface or non-display and the correctness of their answers to the experiment tasks.

Another work is by [Pretorius et al. 2010]. The assumption that they have and prove through their work, is that eye tracking for usability only makes sense, when they are testing interfaces for non-experts. Their use of eye tracking in this context was, to determine what kind of cognitive operations of participant happened. They conducted a study with eye tracking and a usability testing observations, which was a variation of the thinking aloud protocol. The results showed that participants had unnoticed usability problems, which would not have been detected without eye tracking. They conclude, that experts would notice the usability problems in the think aloud protocol and thus their assumption is proven. In comparison, we did not use a think aloud protocol or any other usability testing observations except the eye tracking and screen recording.

Conclusions and Future Work

In the following, we will summarize this thesis and present ideas for future work.

8.1 Conclusions

In this thesis, we researched requirements for experiment management systems and developed an approach for improving the usability of the ExplorViz experiment mode and enhancing it with the features of tracking the eyes and recording the screen during an experiment. We implemented our approach and conducted an experiment to evaluate and test the functionality of our implementation. With the experiment, we confirmed the functionality. We analyzed the experiments results and we found data, which implies that the correctness of our developed tasks correlates with the amount of experience the participant has with ExplorViz. There was also data we found, implying that the percentage of the amount of time a participant looks at the question, the interactive interface, and not at the display, differs between participants. Our experiment population was small and we could not correlate any more eye tracking results. But the eye tracking data and screen recording media files contain versatile information. Thus, we summarize that they do enhance the experiment mode.

8.2 Future Work

During the development of our approach, we and others providing feedback, obtained ideas which would improve the ExplorViz experiment mode. We will present them in the following.

During the set-up of the our experiment, we modified the legacy interactive tutorial of ExplorViz interface. The steps are saved in a configuration and for each step, there is a modal with text displayed. For each modal exists one text file with the text for the modal which is located in a specific folder on the ExplorViz server. Changing the steps and the texts was tedious and complicated, because the order of the text files was set with their file names. An idea would be to save the texts in a similar way to the experiments, as JSON file on the server.

Then we mentioned the bug we experienced during the experiment, where the screen recordings were lost. It is fixed, but now there are, as mentioned in Chapter 5, parts of the

8. Conclusions and Future Work

media file. An automated process where the media files gets merged together would be very useful.

During our analysis of the experiment data, we noticed an important feature that would be really useful. The possibility to export the experiment data or download it is implemented. There exists also the option to upload a single experiment file. But there is no possibility to upload experiment results. We might not always have access to the ExplorViz server, especially when it might only be in temporary use, to conduct an experiment. Thus, in our case it was not possible without a workaround to access the experiment data and replay the video and eye tracking data in the browser.

In Chapter 6 we mentioned, that we used a script to compute results of the eye tracking data. And that we used a vague method to enter the moment when and where the question modals were moved. It would be very useful if the computing could be automated. The results would be obtained easier and more precise. And there is potential in the eye tracking data, for example if we would analyze the interface with a heatmap, determining where participants looked at the most. And with a greater population including professional users, the results might be more distinct.

Bibliography

- [Basili et al. 1986] V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in software engineering. *IEEE Transactions on software engineering* 7 (1986), pages 733–743. (Cited on pages v, 1, and 48)
- [CanJS]. *Canjs*. URL: <https://v2.canjs.com/index.html>. (Cited on page 6)
- [Cornelissen et al. 2007] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. Van Wijk, and A. Van Deursen. Understanding execution traces using massive sequence and circular bundle views. In: *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*. IEEE. 2007, pages 49–58. (Cited on page 4)
- [Finke 2014] S. Finke. Automatische Anleitung einer Versuchsperson während eines kontrollierten Experiments in ExplorViz. Master's thesis. Christian-Albrechts-Platz 4, 24118 Kiel, Germany: Christian-Albrechts-Universität zu Kiel, 2014. (Cited on pages 2, 4, 10, 31, and 65)
- [Fittkau et al. 2013] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: the explorviz approach. In: *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*. IEEE. 2013, pages 1–4. (Cited on pages v, 1, 3, and 65)
- [Fittkau et al. 2016] F. Fittkau, A. Krause, and W. Hasselbring. Software landscape and application visualization for system comprehension with explorviz. *Information and Software Technology* (2016). (Cited on page 4)
- [GWT]. *Google web toolkit*. URL: <http://www.gwtproject.org/>. (Cited on page 6)
- [Ioannidis et al. 1997] Y. E. Ioannidis, M. Livny, A. Ailamaki, A. Narayanan, and A. Therber. Zoo: a desktop experiment management environment. *ACM SIGMOD Record* 26.2 (1997), pages 580–583. (Cited on pages 10 and 13)
- [Jacob and Karn 2003] R. Jacob and K. S. Karn. Eye tracking in human-computer interaction and usability research: ready to deliver the promises. *Mind* 2.3 (2003), page 4. (Cited on pages 1 and 7)
- [Jakobovits et al. 2000] R. Jakobovits, S. G. Soderland, R. K. Taira, and J. F. Brinkley. Requirements of a web-based experiment management system. In: *Proceedings of the AMIA Symposium*. American Medical Informatics Association. 2000, page 374. (Cited on pages 10, 11, and 18)

Bibliography

- [Kevic et al. 2015] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz. Tracing software developers' eyes and interactions for change tasks. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: ACM, 2015, pages 202–213. URL: <http://doi.acm.org/10.1145/2786805.2786864>. (Cited on page 65)
- [Likert 1932] R. Likert. A technique for the measurement of attitudes. *Archives of psychology* (1932). (Cited on page 50)
- [Pretorius et al. 2010] M. C. Pretorius, J. van Biljon, and E. de Kock. "Added value of eye tracking in usability studies: expert and non-expert participants". In: *Human-Computer Interaction*. Springer, 2010, pages 110–121. (Cited on page 65)
- [Shadish et al. 2002] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and quasi-experimental designs for generalized causal inference*. Wadsworth Cengage learning, 2002. (Cited on pages 61 and 63)
- [Sharafi et al. 2015] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology* 67 (2015), pages 79–107. (Cited on page 7)
- [Sharif et al. 2016] B. Sharif, T. Shaffer, J. Wise, and J. I. Maletic. Tracking developers' eyes in the ide. *IEEE Softw.* 33.3 (May 2016), pages 105–108. URL: <http://dx.doi.org/10.1109/MS.2016.84>. (Cited on page 7)
- [WebRTC-Experiment]. *Webrtc-experiment project*. Muak Khan. URL: <https://www.webrtc-experiment.com/>. (Cited on page 8)
- [Wettel et al. 2011] R. Wettel, M. Lanza, and R. Robbes. Software systems as cities: a controlled experiment. In: *Proceedings of the 33rd International Conference on Software Engineering*. ACM. 2011, pages 551–560. (Cited on page 4)

Appendix A: Install Instructions

The current state to carry out an experiment with eye tracking and screen recording on a local computer, we have to go through the following different points.

- ▷ Download and install the Chrome Web Browser. We are restricted to this browser because of the chrome extension that we need to record the screen. Install the chrome extension *Screen Capturing* from *Experiment WebRTC*.
- ▷ Install the tobii EyeX Controller hardware and start it.
- ▷ Download the compiled version of the external communications server with all dependencies. Execute it.
- ▷ Download, install and run the ExplorViz server¹. Because of security restrictions of the screen capturing, it does only record when the website uses either SSL or exists on localhost.
- ▷ An Administrator creates an experiment and generates user. He sets the options eye tracking and screen recording to true and starts the experiment.
- ▷ A participant logs into the ExplorViz website. Fills out questionnaire and finishes the experiment.
- ▷ An Administrator can watch the screen recording and the movement of the participants eyes.

When using the chrome extension, we have to be careful during the experiment. There are problems during the recording of the screen if we use two displays. Afterwards, the mouse is not correctly displayed in the recording.

As optional improvement for the screen recordings, we used the popular free software `ffmpeg`² and installed it on the computer. Then we use the command shown in Listing X on the command line to convert a `webm` media file to a `mp4` media file.

Listing 1. Command to Transform `webm` to `mp4`

```
1 ffmpeg -i userX.webm userX.mp4
```

¹<https://github.com/ExplorViz/Docs/wiki/Installing-and-configuring-software-for-developers>

²<http://ffmpeg.zerance.com/builds/>

. Appendix A: Install Instructions

Table 1. Postquestions Mean and Standard Deviation

Concern	Version
Google Chrome	58.0.3029.110 (64-bit)
Screen Capturing	3.4
Eye tracker model	EyeX Controller
Eye tracker serial number	EYEX2-030145638033
Firmware version	2.0.2-33638
Tobii EyeX Controller Core version	2.0.9
Tobii eyeX controller driver version	2.0.4
Tobii Service version	1.9.4.6493
Tobii EyeX Engine Version	1.9.4.6493
Tobii EyeX Config version	3.2.9.521
Tobii EyeX Interaction version	2.1.1.3125

Appendix B: Informed Consent

Informed Consent

Thank you very much for taking part in this experiment. You can stop and cancel your participation anytime. Your data will be used anonymously.

The goal of this experiment is to determine whether eye tracking adds a value for studies with interactive interfaces, in particular ExplorViz. ExplorViz is a web-based tool for visualizing large software landscapes.

The study consists of three parts. In all three you will have to answer questions. The first and third part are statistical questions about your person and your experiences during the experiment. The second part will take up most of the time and requires interacting with the visualization interface of ExplorViz. During that time, your eyes will be **tracked** and the screen will be **recorded**.

If you have any questions, ask your attendant anytime.

Appendix C: Raw Experiment Results

Participant	UserID	Question Modal %	Interface %	Non-Display %	questionModal in ms	interface in ms
1	7	14,3342315350455	77,215637503376	8,450130961578	228.211	1.229.327
2	8	23,1395246446273	75,6698237970658	1,1906515583068	281.176	919.489
3	9	25,9129100988462	71,6913415576514	2,3957483435024	312.199	863.738
4	10	11,9141677141318	71,3721748592287	16,7136574266394	88.909	532.612
5	12	6,0537035292429	89,8825305393680	4,0637659313890	79.811	1.168.944
6	13	21,5171526416579	75,8792365421652	2,6036108161768	314.723	1.109.856
7	14	14,6661919792295	78,3376677506327	6,9961402701377	168.901	902.164
8	15	21,2752656803500	73,5908522608876	5,1338820587622	245.040	845.595
9	17	8,7807037024131	85,8568328138579	5,3624634837290	168.833	1.650.832
10	18	15,3089221248066	79,8195258411864	4,8715520340069	244.893	1.276.853
11	20	16,8448387276919	74,3133727439295	8,8417885283786	253.374	1.117.795
12	21	21,4703282312954	76,9764824595377	1,5531893091668	247.245	886.435
13	22	5,0858530546164	94,5384505499738	0,3756963954097	53.404	992.701
14	24	32,1263247008663	67,5711223464714	0,3025529526621	544.937	1.146.163
15	26	17,5890391574318	77,8184989730503	4,5924618695179	258.539	1.143.844

. Appendix C: Raw Experiment Results

nonDisplay in ms	fullTime in ms	T1.1 Score	T1.2 Score	T1.3 Score	T2 Score	T3 Score	T4 Score	Experience JPetStore (1-5)
134.532	1.592.070	1	2	0	0	2	1	2
14.468	1.215.133	1	2	0	1	2	2	1
28.864	1.204.801	1	2	0	2	2	2	2
124.725	746.246	0	0	0	0	0	0	1
49.671	1.298.426	1	0	0	2	2	2	1
38.082	1.462.661	1	2	1	3	2	0	1
80.570	1.151.635	1	0	0	2	2	0	2
59.130	1.149.765	1	2	1	1	2	0	1
103.108	1.922.773	1	2	1	2	2	2	1
77.929	1.599.675	1	2	1	2	2	2	1
132.995	1.504.164	0	1	1	2	2	1	1
17.886	1.151.566	1	2	1	2	2	2	2
3.945	1.050.050	0	0	0	1	2	2	1
5.132	1.696.232	1	2	1	2	2	0	1
67.504	1.469.887	1	0	1	2	2	0	1

. Appendix C: Raw Experiment Results

Experience ExplorViz(1-5)	Pressure time (1-4)	Pressure eye tracking (1-4)	Pressure Screen Recording (1-4)	Performance ExplorViz (1-4)	Difficulty T1.1	Difficulty T1.2
1	2	1	1	3	1	2
1	1	1	1	1	2	2
1	1	1	1	4	2	2
1	2	1	1	2	2	3
2	2	1	1	3	1	2
3	1	2	2	3	3	2
2	3	1	2	2	2	2
4	1	2	1	3	2	2
1	2	1	1	4	2	2
1	2	1	1	3	2	1
1	3	1	2	2	2	2
3	1	1	2	3	2	2
1	1	1	1	2	2	2
4	3	1	2	3	2	2
2	1	2	3	3	2	1