Informatics

Visualizing Software Repository Evolutions in Software Engineering Education

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Daniel Kaufmann, BSc
Matrikelnummer 01639360

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Thomas Grechenig

Wien, 13. Jänner 2026

Unterschrift Verfasser	Unterschrift Betreuung

Informatics

Visualizing Software Repository Evolutions in Software Engineering Education

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Daniel Kaufmann, BSc
Registration Number 01639360

to the Faculty of Informatics

at the TU Wien

Advisor: Thomas Grechenig

Vienna, 13th January, 2026

Signature Author

Signature Advisor

Visualizing Software Repository Evolutions in Software Engineering Education

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Daniel Kaufmann, BSc
Matrikelnummer 01639360

ausgeführt am
Institut für Information Systems Engineering
Forschungsbereich Business Informatics
Forschungsgruppe Industrielle Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuung: Thomas Grechenig

Wien, 13. Jänner 2026

Erklärung zur Verfassung der Arbeit

Daniel Kaufmann, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. Jänner 2026

Daniel Kaufmann

Danksagung

Ich möchte meinen Dank all den Menschen aussprechen, die mich während dieser Reise des Diplomarbeitsschreibens unterstützt haben. Das wären meine Familie und meine Partnerin, die mich immer davon abhielten, aufzugeben, wenn ich schon kurz davor war. Meine Freunde, die mich motiviert haben, weiterzumachen. Meine Betreuer, die all meine Fragen beantwortet haben, und geduldig waren, wenn mal etwas ein wenig länger gedauert hatte als erwartet. An euch alle - ich danke euch! Ohne euch wäre diese Arbeit nie abgeschlossen worden.

Acknowledgements

I would like to thank all the people who continuously supported me during the journey it has been to write this thesis. These would be my family and my partner, who always told me to never give up when I was about to do so. My friends, who kept motivating me to keep going. My supervisors, who answered all my questions and were patient with me even when something took a bit longer than expected. To all of you - thank you! This thesis would never have been completed without you.

Kurzfassung

Tutoren und Universitätsassistenten, die Software-Engineering-Kurse betreuen, fällt es oft schwer, sich einen Überblick über die von ihnen betreuten Software-Gruppenprojekte zu verschaffen. Informationen über die Projektstruktur, die Häufigkeit von Änderungen im Code oder die Zusammenarbeit zwischen den Teammitgliedern zu bekommen, ist oft mit großem Zeit- und Arbeitsaufwand verbunden. Im Zuge dieser Arbeit wurde ein interaktiver Prototyp entwickelt, der die Entwicklung der Verzeichnis-Strukturen von Software-Repositories visualisiert, mit dem Ziel, diesen Aufwand zu verringern. Dazu wurden mehrere Visualisierungsmockups entworfen, bewertet und ein interaktiver, webbasierter Prototyp entwickelt. In einer Mixed-Method-Within-Subject-Evaluierung wurde diese Implementierung mit einem state-of-the-art-Tool, IntelliJ IDEA, in Bezug auf die Effizienz der Analyse von Software-Repositories verglichen. Die Ergebnisse deuten darauf hin, dass der Visualisierungsprototyp zum Teil nützlicher ist als IntelliJ IDEA. Insbesondere in den Bereichen der verbesserten Projekttransparenz und der allgemeinen Benutzerfreundlichkeit scheint der Prototyp effizienter und effektiver zu sein, während die Ergebnisse für die Kategorie Refactoring-Analyse entweder eine geringere Nützlichkeit oder bezüglich der Häufigkeit von Änderungen im Projekt einen weniger eindeutigen Unterschied zeigen. In der Praxis wäre es mit der entwickelten Visualisierung möglich, Universitätsassistenten oder Tutoren bei der Analyse und Benotung von Softwareprojekten von Studierenden zu unterstützen. Um die präsentierten Ergebnisse zu bestätigen, werden zusätzliche Evaluierungen mit einer größeren Anzahl an Teilnehmern empfohlen.

Keywords: Software Visualization, Software Engineering Education, Software Repository Mining, Git, IntelliJ, Storyline Visualization

Abstract

Tutors and university assistants in software engineering courses often have a hard time getting an overview of the student group projects they supervise. Information about project structure, frequency of changes in the code and collaboration between team members can be hard to come by efficiently, leading to significant time and effort spent on analyzing those projects. To address this, an interactive prototype was developed to visualize the evolution of the directory-tree of software repositories with the goal of reducing this effort. This involved designing multiple visualization mockups, evaluating the selected one, and developing an interactive web-based prototype. In a mixed-method within-subject evaluation, this implementation was compared against a state-of-the-art tool, IntelliJ IDEA, regarding the efficiency of analysing software repositories. The results indicated that the visualization prototype was partially more useful. Specifically, the areas of project transparency and overall usability outperformed IntelliJ IDEA, while the results for refactoring analysis indicated lower usefulness, and the frequency-of-changes analysis showed a less clear or inconclusive difference. In practice, it would be possible to use the proposed visualization to support university assistants or tutors when analysing and grading student group software projects. Future evaluations with a larger participant size are recommended to confirm these findings.

Keywords: Software Visualization, Software Engineering Education, Software Repository Mining, Git, IntelliJ, Storyline Visualization

Contents

CHAPTER 1

Introduction

Analyzing the work done on a student group-project in a short amount of time is a challenge that student assistants and tutors face often. It is not an easy task to transparently understand what happened over time during the development of a software project, especially with regard to individual contributions [40] [29] [23] [35]. Persons that are trying to get an overview of a software project, such as tutors analyzing a student project, often have a hard time getting information about changes in the project-structure, frequency of changes in the code and collaboration between team-members. Thus, they have to spend a lot of time and effort researching and analyzing past commits to acquire this knowledge [60] [48].

This thesis is concerned with the development of a prototype which visualizes the evolution of the directory-tree of a software repository in a comprehensible way for educators in order to address this. For this reason, the academic contribution of this thesis is part of the areas of Mining Software Repositories (MSR) and the Software Visualization.

1.1 Motivation

The main motivation for this contribution is that, while software visualization techniques have been studied in general, gaps still exist [56] [8] [10]. Understanding the architecture of software projects is considered essential for the evolution of software systems [71] and the need for more insights into the evolution of such projects is still there [32].

According to Hojelse et al., „One of the most powerful conceptual tools for managing complexity is the hierarchical organization of the source code of such systems.“ [33]. Only a few approaches seem to tackle the hierarchical directory-level structure or its evolution [67] [33]. Currently existing tools such as Git or IntelliJ IDEA also do not directly support visualizing software projects this way. One aspect of software repository evolution, the detection of the movement of files using Distributed Version Control Systems (DVCS) such as Git, remains a challenge [13] [10].

Especially in an educational setting, student assistants want to be able to see what work was done over the course of a project in order to grade students in a fair way based on their contributions [60]. In group projects, students often do not contribute equally [48] [44] and their perception of effort is often not objective [30]. To achieve a fair grading process, which is one of the most important factors in group project grading schemes [29], we argue that the instructors need to be provided with as much information about a project as possible, while minimizing the required time for evaluation as it can be a time-consuming task [48] [43] [26].

1.2 Expected Results

The aim of the thesis is to make software development processes in education more transparent by visualizing the history of directory trees in a comprehensive way. The expected result is an interactive visualization prototype which satisfies information needs such as being able to see the history of an individual file or directory while switching between different abstraction levels. Furthermore, the prototype should provide useful insights on the development process of an application and highlight frequently changing areas of code and display information about which persons contributed which code. The opposite of frequently changing code — dead code — should also be made visible.

Based on the challenges described in Section 1.1, especially the need for software project evolution comprehension in educational settings in order to achieve fair and transparent grading, this thesis defines three research questions. Each of the research questions targets a specific step in achieving that goal: finding and assessing suitable visualization methods, validating their usefulness through expert feedback, and comparing their effectiveness with existing tools. Combined, the questions attempt to ensure that the proposed visualization directly addresses both the technical and educational needs described above:

1.2.1 RQ1: What are different methods to visualize the evolution of a software repository?

During the first phase of the thesis, research was done to assess existing visualization methods and develop mockups for displaying the history of software repositories. One of these developed prototypes was then chosen for further evaluation.

1.2.2 RQ2: How do domain experts agree with the chosen visualization approach?

The second phase consisted of conducting interviews with experts with the aim of assessing and gathering feedback on the previously chosen visualization method.

1.2.3 RQ3: How does the visualization compare to the state-of-the-art with regard to the following benefits: student project transparency, refactoring frequency, stale code detection, code hotspots?

This research question was concerned with the development of a prototype for visualizing the history of the directory tree of software repositories. This prototype was compared with IntelliJ IDEA to determine its effectiveness with regard to the proposed benefits of student project transparency, refactoring frequency, stale code detection and code hotspots.

1.3 Structure

The structure of this thesis is the following: Chapter 2 provides the theoretical background on Version Control Systems, Mining Software Repositories, data visualization techniques, software project comprehension and the educational context. Chapter 3 looks at related work with regard to repository visualization and how it is used in educational settings. Chapter 4 presents visualization concepts that served as candidates of which one was chosen and the reasoning for selecting the implemented prototype candidate. Chapter 5 describes the implementation of the visualization prototype, while Chapter 6 goes into the evaluation process of the chosen visualization and the results of the performed within-subject mixed-method evaluation. Chapter 7 interprets the presented results, highlights the threats to validity, and provides potential future work. The final chapter, Chapter 8, summarizes the contributions and main results of this thesis.

1.4 Methodology

In order to answer the research questions defined in Section 1.2, the research presented in this thesis followed a multi-step approach. The methodology was a combination of exploring the concept itself, developing a prototype, and evaluating it.

1.4.1 Research Design

The research was done in five phases, which are described in the following subsections.

Literature Review and Concepts

In the first phase, the current literature and theoretical foundations were reviewed (Chapter 2) and related work for software repository visualizations and educational applications explored (Chapter 3).

Concept Development

After exploring visualization concepts, three candidate mockups (storyline, treemap and radial tree) for displaying directory and file evolutions were created (see Chapter 4). The storyline visualization was chosen based on the criteria described in Section 4.2.

Concept Evaluation

The chosen visualization was then validated in an empirical evaluation. The mockups of that visualization were evaluated together with three professionals, as presented in Section 4.3.

Prototype Development

Based on the evaluation of the chosen visualization as described in the previous step, a prototype application was developed in an iterative approach with a modern web technology stack. In the frontend, D3.js was used to render the interactive storyline visualization. The backend was implemented using Node.js and used to mine Git repositories and provide structured JSON data that represented the file operations. Feedback from the initial mockup evaluation was used to refine features such as the author-mode visualization and the navigation. Further details about the implementation can be found in Chapter 5.

Prototype Evaluation

The developed prototype was then evaluated in a second, larger empirical evaluation. This evaluation was designed with a within-subject mixed-method approach with six professionals, which compared the prototype against IntelliJ IDEA (see Chapter 6).

1.4.2 Data Analysis

The quantitative evaluation used the following metrics:

- **Task success rate** (success, partial success, failure),
- **Completion time** per task,
- **Perceived difficulty** (5-point Likert scale),
- **System Usability Scale (SUS)** scores.

The statistical analysis of the gathered data was done using paired t-tests, effect sizes (Cohen's d_z) and confidence intervals. This was done according to recommendations in usability research [68] [46]. The qualitative feedback was thematically analyzed and grouped in order to complement the quantitative findings.

CHAPTER 2

Foundations

This chapter is concerned with introducing the reader to the most important concepts and foundations needed to understand the subsequent chapters of this work.

2.1 Version Control Systems

Version Control Systems (VCS) are tools that enable software development teams to manage changes to source code over time in an efficient way [79]. They make it possible for multiple developers to work on the same project in parallel, while tracking progress systematically. Important features of VCS include the ability to maintain a history of changes, usage of branches and merging and conflict resolution mechanisms [13] .

2.1.1 Source Code Control Systems

Early versions of VCS were primarily local, with one example being Source Code Control Systems (SCCS), which were introduced by Bell Labs in the 1970s. These systems stored version histories on the machines of individual developers [65].

2.1.2 Centralized Version Control Systems

With growing complexity and scale of software projects, the limitations of this approach became apparent, and Centralized Version Control Systems (CVCS) such as Concurrent Versions System (CVS) and Apache Subversion (SVN) were developed. Instead of locally stored version histories, the concept of a central repository was introduced, which provides a single source of truth for developers to check out from and commit changes to [17]. There is potential risk involved with using these centralized systems as users only have the latest file versions on their machines [79].

2.1.3 Distributed Version Control Systems

As a successor, Distributed Version Control Systems (DVCS) were introduced, which addressed these issues by having every developer's working copy act as a repository with full version history. From this, systems like Git and Mercurial emerged, providing fault tolerance and supporting more dynamic workflows, e.g. feature branching and integration. This distributed way of working also enabled offline working and enhanced collaboration through mechanisms like pull requests and code reviews [13]. In the Open Source Software (OSS) space, many projects have been moving from CVCS to DVCS systems [63]. DVCS have certain architectural elements such as commits, branches and merges. We will focus on Git in the following sections, as it is a widely used VCS [18].

Commits

Commits are snapshots of a project at a given time within the DVCS, where the state of the files and the changes that were made are stored. A commit consists of its identifier (the hash of the commit object), a commit message describing what was changed, and metadata such as the author, timestamp, and parent commits. This information enables developers to track and understand the entire history of a project and go back to previous states [13].

Branches

This mechanism allows developers to create a separate line of development in order to work on features or bug fixes without interfering with the other collaborators, which can later on be merged back into the main branch. This capability is an important factor of enabling parallel development and adding contributions from multiple developers [80]. Branches are lightweight and therefore resource-efficient [13].

Merges

When merging, changes from different branches are integrated into a single branch. For example, once development on a branch is ready to be added to the main line of development, it is merged [80]. Depending on the circumstances when merging, different types of merges occur. These include fast-forward merges, where the pointers of a branch are simply moved forward. Another type of merge is the three-way merge, in which a common ancestor is used to reconcile changes from diverging branches [13].

Git Object Model

Git stores repository data in a content-addressable object database. Content is stored as an object under `.git/objects`, identified by a 40-character long SHA-1 hash of the content combined with a header. This makes Git a simple key-value store at its core, on top of which the VCS user interface is built [13] [15].

There are four object types defined by Git [13]:

- **Blob**: This object type that stores the raw file contents.
- **Tree**: A directory-like structure that lists entries, where each entry points to either a blob (file) or another tree (subdirectory). Trees organize blobs into hierarchical snapshots.
- **Commit**: Assigns a top-level tree to metadata such as author, committer, timestamps, message) and to zero or more parent commits, forming the project history as a DAG.
- **Tag**: An annotated, immutable object that points to a commit but can reference any object type [18]. It contains data such as the tagger, a date, a message and a pointer. There are also lightweight tags that are simply references that never move. Annotated tags are more complex as in that they create a tag object and a reference to point to it instead of a commit [13].

References

References are human-readable names stored under `.git/refs` that map to object IDs, moving a branch simply updates its reference. `HEAD` is a symbolic reference (a reference to another reference) that always points to the currently checked-out branch [13].

2.1.4 Further utilization of DVCS

Modern DVCS systems offer features such as rebasing, automatic merging and interactive staging [13]. In order to further improve productivity and ensure high quality software, several techniques can be employed.

Continuous Integration

One paradigm that emerged from Extreme Programming practices called Continuous Integration (CI) is often utilized. It places a great emphasis on integrating changes made during development as often as possible, meaning that the entire application is built and tested on every change [34]. CI is widely adopted, growing, and associated with measurable benefits [31].

2.2 Mining Software Repositories

MSR is a field that is concerned with analyzing the data that can be obtained from software repositories in order to discover information about software projects [39] [28]. This information can be used to understand and improve the software development process with regard to productivity and quality [41]. There are different areas in the field that researchers are exploring, including software evolution, models of software development processes, the characterization of developers and their activities, and the prediction of future software qualities. They also investigate the application of machine learning techniques on software project data, software bug prediction, the analysis of software change patterns, and the analysis of code clones [77].

In comparison to the field of data mining, MSR is a more specialized area as the gathered information can only be understood with domain knowledge in software engineering. It can also be compared to reverse engineering in regards of finding structures and patterns. Reverse engineering is concerned with understanding software by analyzing a snapshot of code. In MSR, researchers take into account sequences of these snapshots in order to comprehend data changes made over time and their analysis often also includes bug reports, the social networks of developers and design documents, which makes the amount of mined data much more large and complex [39].

2.2.1 Processing

Jung et al. describe MSR as being very similar to data mining, which means that the processes behind both are also closely related [39]. Their chart on MSR (Figure 2.1)

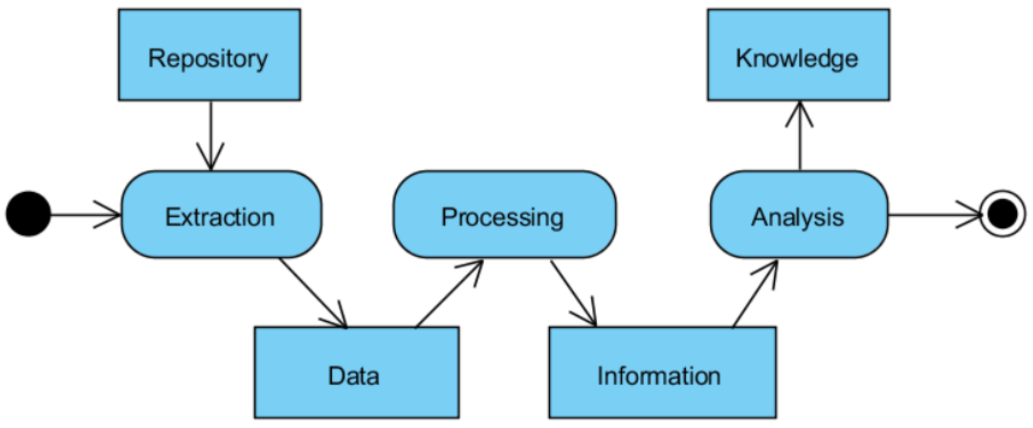


Figure 2.1: The general process of MSR [39]

supports this claim by visualizing objects and processes and how they are connected to each other. In the beginning, data is extracted and classified by the types of the repositories [39]. This classification is necessary as data often comes from various sources, the majority being SCCS such as SVN and Git, bug tracking systems [39] [41] and Stack Overflow [74].

After mining data from SCCS, the extracted source code usually is a set of text strings. In order to process this data on a token-level, Abstract Syntax Trees (AST) are often used. Performing operations on these data structures enables a more accurate analysis of dependencies and structure of the code, but can be more complex due to the amount of tokens and edges involved. For the latter reason, this technique is often only used in specific areas of concern and not all of the data [39].

Analysis

The information gained from processing mined data can then be utilized for bug-fixing [78], change-related activities, development, comprehension enhancement in source code [59] or identification of technical debt [41]. A very common technique to analyze such data sets mined from software repositories is data visualization [16] [9] [57] [42].

Limitations in MSR

Renames are not recorded in Git explicitly. The detection of such is done using heuristics and based on content similarity at diff time. This can miss logical renames, file splits or merges or large-scale refactorings. Approaches based on Abstract Syntax Trees differencing improve on this limitation [21].

2.3 Data Visualization

In order to represent big sets of data mined from software repositories, several visualization techniques can be utilized [5]. These are crucial for converting textual data into visual representations [73] in order to understand trends, see patterns and identify anomalies that occurred during the development of a software project. This section will introduce the reader to some of the most commonly used repository-mining visualization techniques, such as time-series, treemaps and radial trees.

2.3.1 Time-Series

Time-series visualizations, for example storylines [3], are able to show large data sets, such as the evolution of mined software repositories in a more understandable way [22]. As can be seen in Figure 2.2, they enable visualization of changes across many items in just one view, which is an advantage when concentrating on the dependency of time and data. Also, they allow for visual comparison of data at different points in time [4].

This effective way of visualizing data is employed by existing approaches [58] [45] [67] which are investigated closer in Chapter 3. The prototype that was chosen and developed over the course of this thesis also utilizes this visualization technique.

2.3.2 Treemaps

Treemaps are a space-nested layout, which is a space-filling technique used for conveying hierarchical structure while efficiently utilizing screen space. Nodes are shown as rectangular boxes on a 2-D display, which are then divided into further rectangles to represent the child nodes [37]. The disadvantage of this structure lies in its inability to portray its hierarchical structure [73].

By filling the available screen space completely, treemaps provide a compact and comprehensive view of hierarchical data (see Figure 2.3). This makes it useful especially

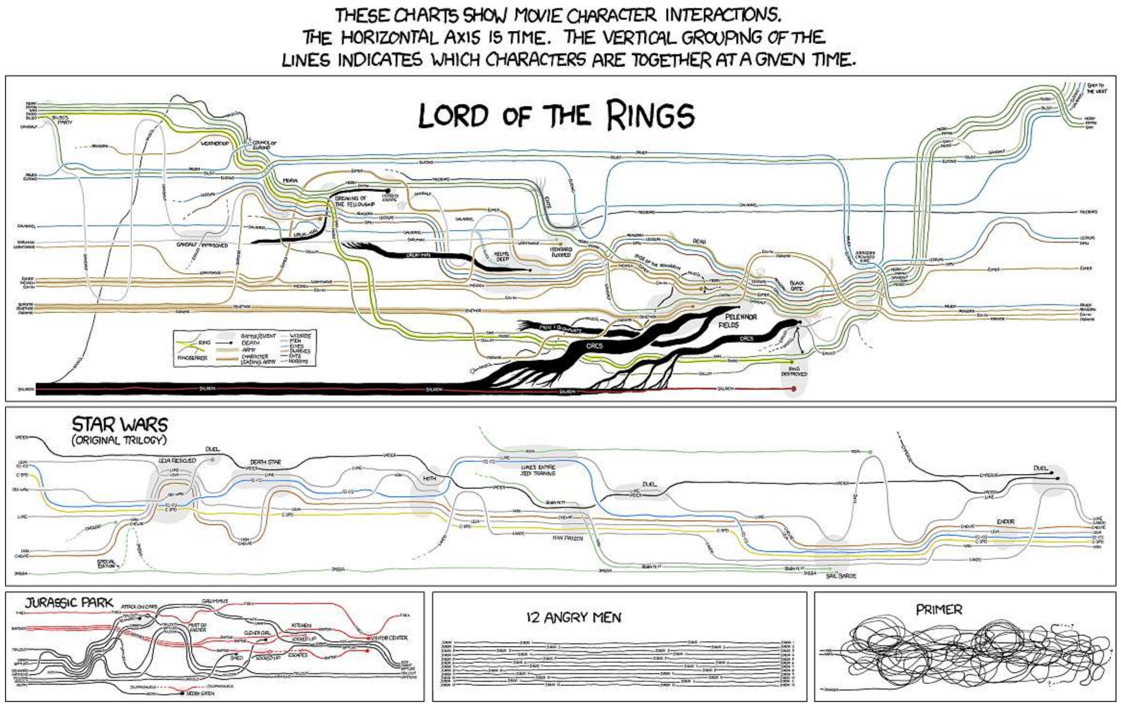


Figure 2.2: An example of a storyline in the form of movie character interactions [88].

when dealing with large datasets. The usage of color-coding to represent categories or attributes is the most important property of this visualization and can provide further insights into complex information. As the human perception is limited in understanding the complexity of graphic representations, the user should be provided with some kind of interactive control over the visualization [37].

In order to overcome the limitations in displaying hierarchical structure, Lü and Fogarty propose *cascading* treemaps which utilize a depth effect and natural padding to use less space [47].

2.3.3 Radial Trees

In a radial tree layout, the root of the tree is placed in the middle of concentric cycles, as can be seen in Figure 2.4. The children of a sub-tree are then distributed into a circular shape, depending on its depth, recursively. This way, space is used more efficiently than it would in a classical tree layout. The downside of this technique is that it is harder to grasp, as its root is harder to find. The ability of tree structures such as the radial tree to visualize any hierarchical structure makes them a common candidate for visualizing the structure of complex software systems [73].

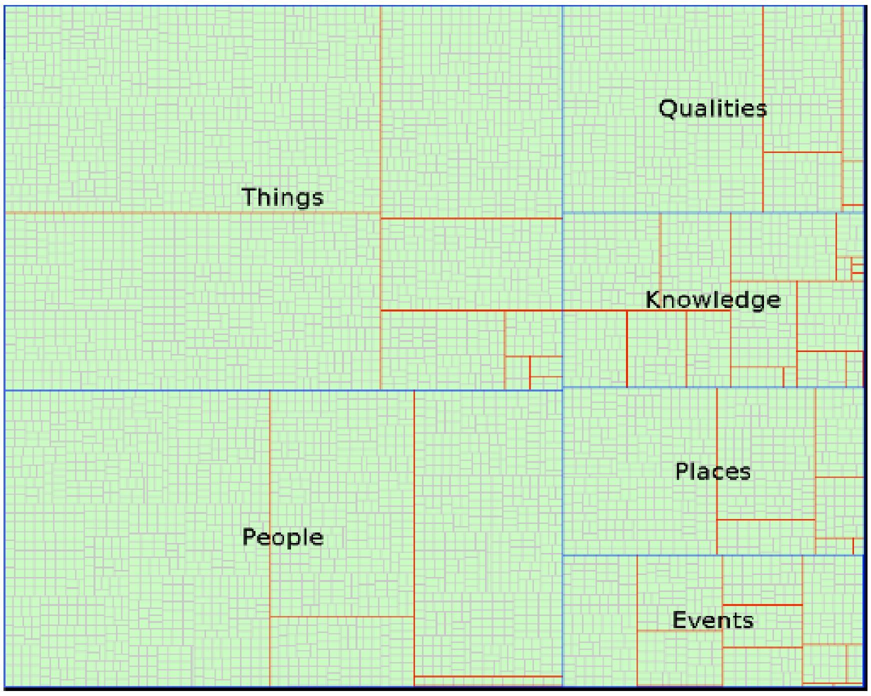


Figure 2.3: An example of a treemap [73].

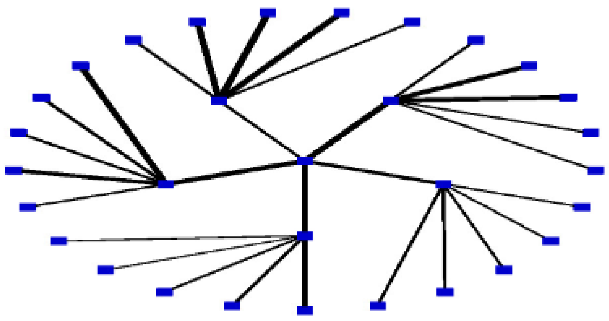


Figure 2.4: An example of a radial tree layout [73].

2.4 Software Project Comprehension

In order to effectively maintain software and drive its evolution, it's essential to be able to comprehend it [2]. There are two classic theories regarding how program comprehension works [62]. The *top-down theory* says that when software developers try to comprehend software, they first make hypotheses, which are then either confirmed and then retained, or rejected and discarded based on the found evidence, the „beacon“, within the analyzed source code [12]. The second theory is called the *bottom-up theory* and builds comprehension on the concept of chunking of parts of the code. These chunks have a certain name and meaning and consist of smaller chunks. The understanding of software developers is built by combining these chunks into increasingly larger chunks. Both theories can also be combined when theorizing software project comprehension [75].

Brooks originally described these mechanisms at the source-code level [12]. Later work by Rajlich and Wilde showed that the same cognitive processes also apply across different abstraction levels, such as domain concepts and high-level design concepts. These concepts are described as the „fundamental building blocks of human learning“ [62].

This extension implies that software comprehension is not limited to line-by-line code understanding, but also includes structural aspects of a system. In this thesis, we theorize that directories, files, and their relationships can themselves act as *chunks* or *concepts*, which developers combine into a coherent mental model of the system's evolution.

2.4.1 Usage in Education

These concepts can also apply in an educational context. Tutors and student assistants often need to focus on the higher-level structures of a student project when analyzing it, such as individual student contributions, how collaboration and project work evolved or how the overall architecture changed over time [48].

This assessment can become challenging if the evaluated project is the result of collaboration within a student group over several months and when has to be done under time-pressure. Lungu et al. claim that comprehension of such cases can be aided by utilizing software visualization tools [48] (also see the discussion on educator information needs and time constraints in Section 1.1 and Chapter 7).

Version-control data can be mined to visualize contributions to be used for grading and feedback [51] [27]. Commit-impact and code-quality trends are utilized to highlight where support and guidance are needed [26] or which branching and testing practices were followed [49]. Other approaches include contribution dashboards to estimate performance from commits [61], pull requests and file activity [60] and line-level activity metrics combined with submission frequency to differentiate between stronger and weaker contributions [55]. Repository signals were also used to generate individualized assessments [64].

Focusing on the visualization of contributions, the work done by Sein-Echaluce et al. shows that providing continuous transparency on individual contributions and error states has a great effect on the ability of students to comprehend their own code [70]. The usage of such tooling was also shown to be accepted and found useful by students participating in group projects [27].

CHAPTER 3

Related Work

This chapter presents related work that is related to the idea proposed in this thesis. It structures this content into two sections. The first focuses on existing visualization approaches, while the second highlights related topics in the educational context of software repository mining.

3.1 Visualization of Software Repository Evolutions

There exist numerous approaches that aim to visualize the evolution of software repositories in order to make it more comprehensible. This section presents a selected subset of contributions in the area of software visualization.

3.1.1 Software Evolution Storylines

Software Evolution Storylines is a contribution that directly utilizes a storyline visualization in order to display the evolution of software projects. Ogawa and Ma visualize the commits of contributors within certain parts of the software repository in a metro-map style where each contributor is a separate „line“ with a unique color [58], as can be seen in Figure 3.1. It focuses on the interactions between developers over the course of a software project, while not considering the hierarchical directory-tree structure [58].

This differentiates the used metro-map style from the storyline-visualization proposed in Subsection 4.1.1, as the latter displays the evolution of directory structures and does not focus on developer interactions themselves.

3.1.2 CodeTimeMachine

The CodeTimeMachine is a plug-in for the IntelliJ IDEA IDE developed by Aghajani et al. and can be used for displaying the underlying commit history of individual files in

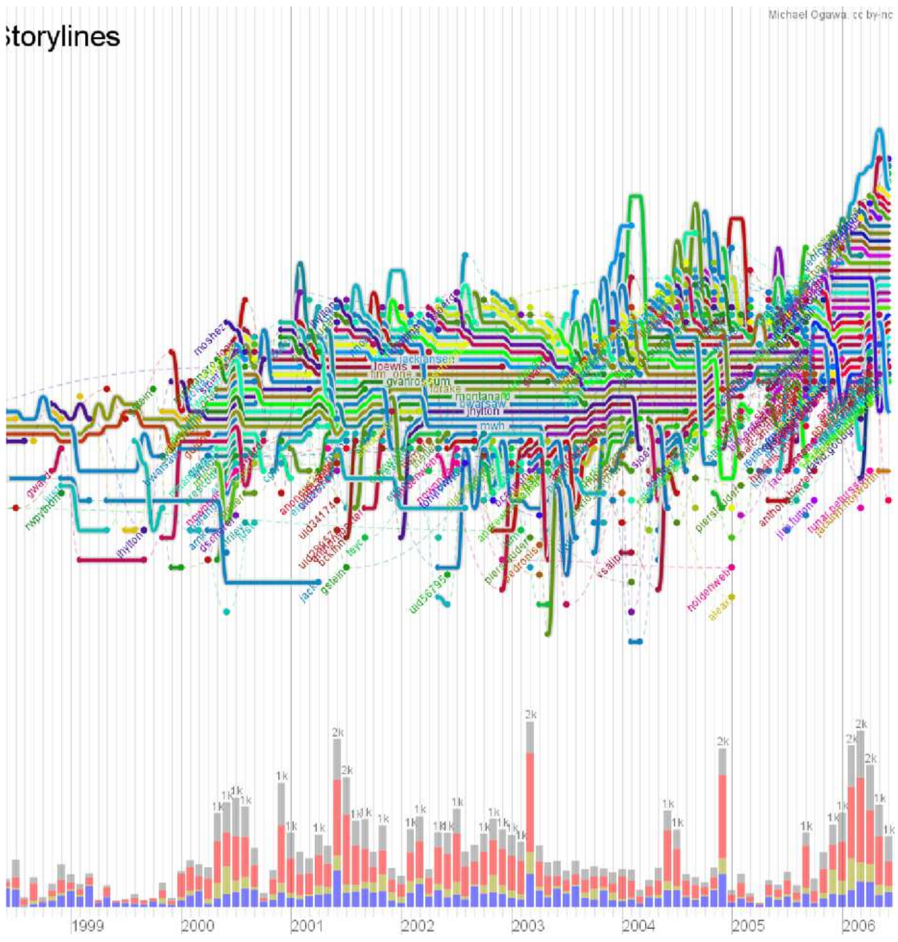
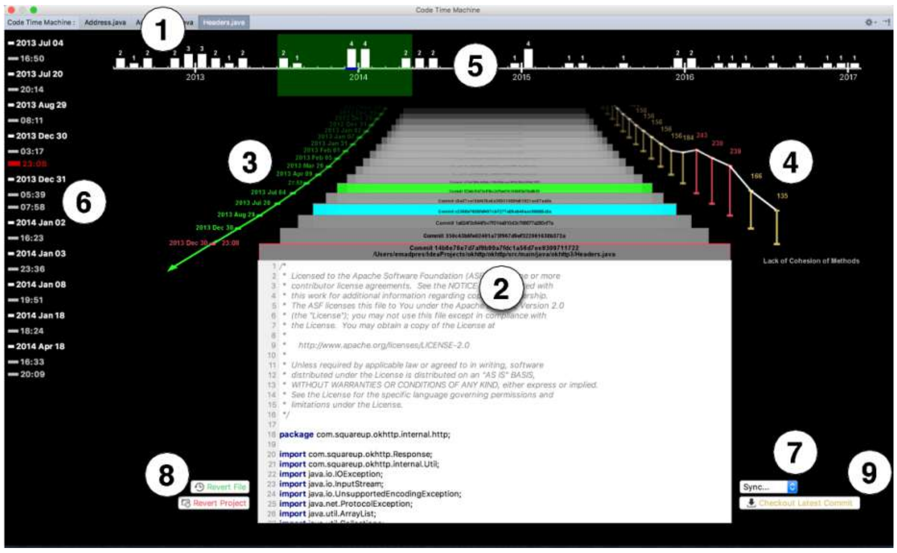


Figure 3.1: One example of the visualization of commits of the Python software repository in *Software Evolution Storylines* [58].

a stack-like manner (see Figure 3.2). The main feature is to be able to browse the history of files to see which changes were made at which point in time. Additionally, metrics such as Lines Of Code (LOC), number of methods, and the cyclomatic complexity are displayed. The range of commits can be limited to narrow down the files commit-history. A diff view is provided which enables users to compare specific versions of a file [1].

While this visualization provides a seemingly useful way to display the history of individual files, it does not provide the user with an overview of the whole project. The user has to select each file separately to be able to comprehend their evolution, which could be time-consuming when not being sure which file to analyze first. The visualization that was contributed in this thesis differentiated itself from this related work by aiming to tackle the need of higher-level structures in software development education, which was discussed in Subsection 2.4.1.



```
Commit 14b6e76e7d7af8b99a7fdc1a56d7ee8309711722
/Users/emadpres/IdeaProjects/okhttp/okhttp/src/main/java/okhttp3/Headers.java
/*
 * Licensed to the Apache Software Foundation (AS           e or more
 * contributor license agreements.  See the NOTICE           ed with
 * this work for additional information regarding co           rship.
 * The ASF licenses this file to You under the Apache           Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.squareup.okhttp.internal.http;

import com.squareup.okhttp.Response;
import com.squareup.okhttp.internal.Util;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.net.ProtocolException;
import java.util.ArrayList;
```

Figure 3.2: The stack-like visualization that is provided by *The CodeTimeMachine* [1].

3.1.3 CodeTimeline

The visualization *CodeTimeline* aims to share the tribal knowledge of development teams in order to make it available to developers new to specific software projects (see Figure 3.3. It does so by visualizing word clouds and specific events that occurred during the life-cycle of a software repository [45].

While this technique provides a high-level overview of the evolution of software projects, it does so by focusing on events that occurred over the course of the project. While this could also be useful in an educational setting, our contribution focuses on a more technical aspect.

3.1.4 RepoGrams

Rozenberg et al. developed a tool called *RepoGrams* which makes the analysis and comparison of software repositories more efficient. Using a visualization that displays each commit as blocks of different length, commits in project repositories can be analyzed to evaluate a software projects evolution, as demonstrated in Figure 3.4 [66].

The metric-based approach enables developers to extend the tool by adding their own metrics. The implemented metrics include the number of directories modified by a single commit, the number of times a file has been modified in the past and the number of files modified in a commit (also new and deleted files) [66].

While these metrics provide an interesting insight into *how many* files were modified with each commit and by whom, it does not provide developers with an overview of *when* and *which* files and directories were modified in the past [66]. The visualization proposed by

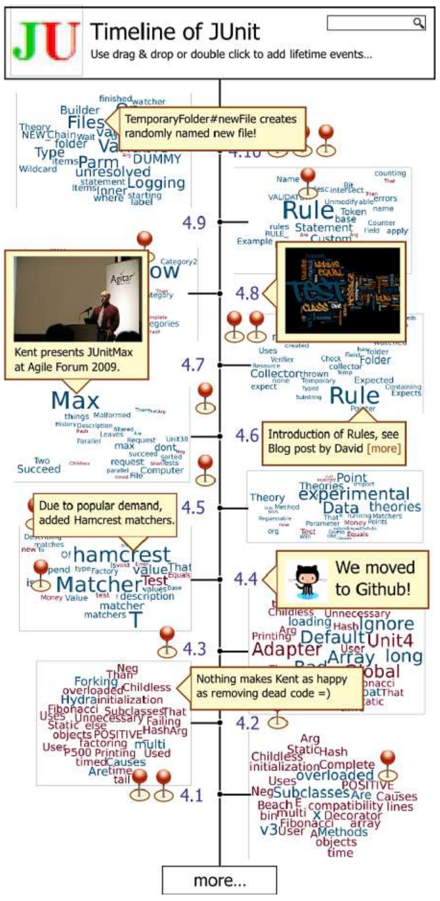

Timeline of JUnit
Use drag & drop or double click to add lifetime events...
TemporaryFolder#newFile creates randomly named new file!
4.9
4.8
Kent presents JUnitMax at Agile Forum 2009.
4.7
4.6
Introduction of Rules, see Blog post by David [more]
Due to popular demand, added Hamcrest matchers.
4.5
4.4
We moved to Github!
4.3
Nothing makes Kent as happy as removing dead code =)
4.2
4.1
more...

Figure 3.3: Visualizing word clouds and specific events in the life-cycle of the JUnit software repository [45].

this thesis aims to achieve this by providing a high-level overview of the evolution history in software repositories.

3.1.5 CodeCity

CodeCity is an often cited project presented by Wettel and Lanza that visualizes object oriented systems as habitable cities. The buildings of the city are placed according to specific rules and display constructs such as packages, classes, methods and their relationships. The size of the base of the buildings depends on the number of methods a construct contains, while the height is determined by the number of attributes (see Figure 3.5) [76].

While this kind of visualization seems to be a good way to provide an overview of the complexity of a software project to developers, it does not give any insights into the actual evolution of repositories' files and directories.

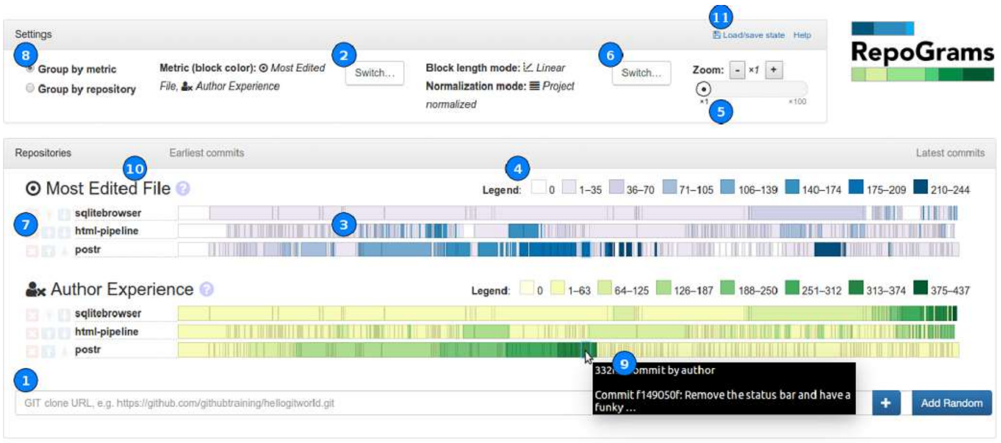

Settings
Load/save state
Help
RepoGrams
Group by metric
Group by repository
Metric (block color): Most Edited File, Author Experience
Switch...
Block length mode: Linear
Normalization mode: Project normalized
Switch...
Zoom: - ×1 +
Repositories
Earliest commits
Latest commits
Most Edited File
Legend: 0 1–35 36–70 71–105 106–139 140–174 175–209 210–244
sqlitebrowser
html-pipeline
postr
Author Experience
Legend: 0 1–63 64–125 126–187 188–250 251–312 313–374 375–437
sqlitebrowser
html-pipeline
postr
332 commit by author
Commit f149050f: Remove the status bar and have a funky ...
GIT clone URL, e.g. https://github.com/githubtraining/hellogitworld.git
+
Add Random

Figure 3.4: *Repograms* provides a metrics-based visualization where commits are displayed as blocks [66]

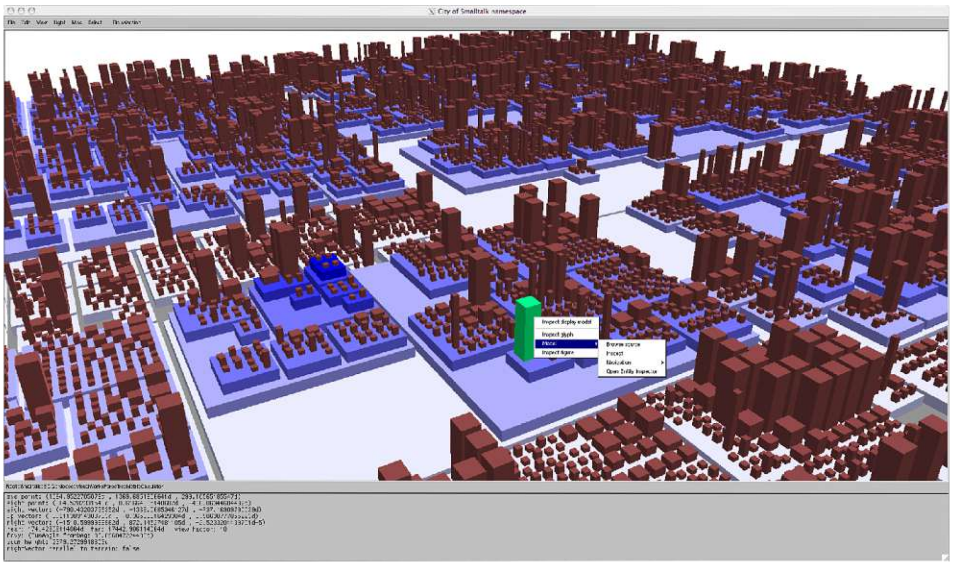


Figure 3.5: The *CodeCity* of a software repository [76]

3.1.6 RepoVis

Another project that deals with giving developers an overview of the structure, evolution and the status of collaborative software projects is *RepoVis*, which was developed by Feiner and Andrews. Files in software repositories are visualized as blocks hanging on a wall with Lines Of Code inside, which are colored depending the chosen metric such as last modification, contributor or file types, as can be seen in Figure 3.6. These visuals can be filtered using a full-text search [20].

In contrast to this contribution, the visualization prototype of this thesis aims to visualize

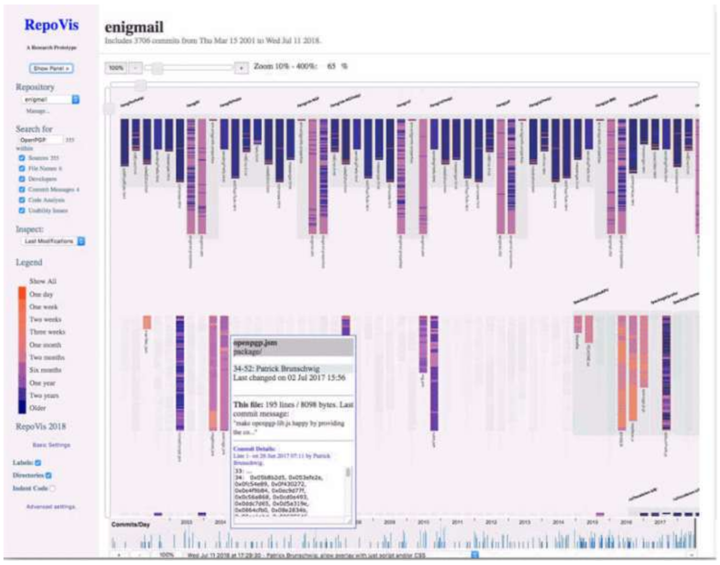


Figure 3.6: *RepoVis* displays files as blocks hanging on a wall [20]

the evolution of a software repository, including movement operations.

3.1.7 Tracking based on AST

Fujimoto et al. present an algorithm for tracking the history of files using Abstract Syntax Trees (AST) without providing any kind of visualization [21]. Further works focus on comparing software products by extracting unified directory trees [67] and creating expressive storyline visualizations using reinforcement learning [72]. Especially the AST tracking is an area of research that inspired the approach of tracking movement operation in the visualization idea of this thesis.

3.1.8 Git-Truck: Hierarchy-Oriented Visualization of Git Repository Evolution

Git-Truck is a tool that enables users to get a visual overview of a Git repository in order to analyze their evolution. Hojelse et al. focus on visualizing the structural information of a software repository [33].

For this, they offer both circle packing and treemap visualizations. Git-Truck provides five predefined visualization methods that highlight different areas of a software project:

- File Extension
- Number of Commits
- Last Changed

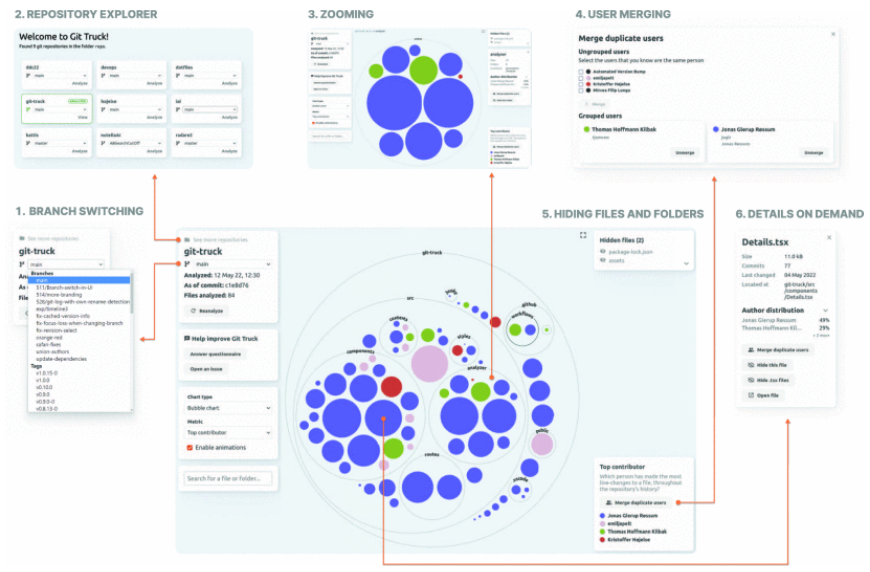


Figure 3.7: An overview of the UI of *Git-Truck* [33]

- Single Authors
- Top Contributors

As shown in Figure 3.7, the tool offers interactions such as switching branches, zooming in and out of folders, merging together users, hiding files and directories and displaying additional details on demand [33].

This approach comes very close to what we wanted to achieve with our contribution, especially with regard to highlighting contributions and numbers of commits in a certain time-frame. Our visualization differentiates itself in that it provides a totally different metaphor of visualizing these things, in the form of storylines (see Section 4.2).

3.2 Education Context

This section explores the related work and state of the art regarding the educational context of software repository mining. Various methods have been explored to assist student assistants and instructors in grading student projects with regard to functional requirements, code quality, complexity, maintainability and determining the contribution of students to team projects.

3.2.1 Mining Software Repositories from Student Projects

Mittal and Sureka present methods to determine students' collaboration on software projects, as well as commit message quality, consistency of commits and component and developer entropy. The latter is used to measure the contribution of each student developer across all components of a project, which should, according to the authors, be evenly distributed in an ideal scenario. Using a radar chart, they measure individual students' contributions to certain modules of software projects. To determine the co-development of developers working on the same components, they define the metrics *Component Entropy* and *Developer Entropy* [51]. To add to this research, the prototype that is developed as part of this thesis aims to extend this related work by measuring developer contribution on a file-level, to enable an even better and more granular comprehension of the collaboration of students to the evolution of software repositories.

3.2.2 Mining Student Repositories to Gain Learning Analytics

Robles and Gonzalez-Barahona explore a different way of determining the original authorship of students' contributions. In addition to automated plagiarism checks, they use software repository mining to generate personalized exams that are supposed to test students' knowledge about the project they developed. These tests consist of code snippets, black-box questions and questions about specific scenarios [64]. The visualization prototype developed in this thesis assumed the authenticity of student contributions, as additional verification methods such as the ones presented in the research of Robles and Gonzalez-Barahona were outside of the scope.

3.2.3 Measuring Students Source Code Quality in Software Development Projects

Hamer et al. analyzed the effect of students' changes on the quality of their projects to identify possible challenges and areas of improvement in software engineering education. For this, they mined 2253 commits of an undergraduate student project and determined the quality metrics such as complexity, maintainability, duplication, and security for each of the changes that were made. Based on that, they calculated how the overall quality was impacted. They found that the students did not practice continuous code integration and suggest that the used metrics can be used to help instructors determine the evolution of students' projects and to identify possible gaps in students' knowledge as soon as possible [26]. This focus on quality and impact of commits could be an interesting enhancement to the contribution analysis features implemented in the prototype visualization of this thesis (see Section 7.5).

3.2.4 Assessing Software Quality of Agile Student Projects by Data Mining Software Repositories

Koetter et al. developed a tool that extracts Git logs from six student software repositories and performs an analysis that determines identifying contributing factors to the success or failure of such projects. The main areas of interest consisted of modularity, re-usability, maintainability and testability of the software. They consisted of metrics and factors such as coupling between objects, message passing coupling, data abstraction coupling, weighted methods per class, documented public API, depth of inheritance tree, number of methods and polymorphism factor. Furthermore, they conducted a survey using questionnaires for the instructors who coached the projects regarding the quality of the analyzed projects. Their recommendations for software development based on the results, among others, are to keep the workload evenly distributed throughout the project and to limit the number of participants per student team [43]. Similar to the work of Hamer et al., this work could also be a valuable future addition to our visualization prototype, which Section 7.5 explores.

3.2.5 Using Process Mining for Git Log Analysis of Projects in a Software Development Course

Macak et al. analyzed the software development behaviors of students using software repository mining of student projects. They present a methodology for identifying which features to extract during the mining process and describe the way the data should be processed to be used for analyzing the development practices of students. The authors found that the majority of students did not practice Test-Driven Development (TDD), were inconsistent in the usage of branches, did not distribute their work evenly and did not contribute to their teams the same way [49]. Especially the uneven contributions to student projects motivated the contribution analysis features in our proposed visualization (see Section 4.1.1).

3.2.6 Extracting New Metrics from Version Control System for the Comparison of Software Developers

Moura et al. present additional metrics to compare the contributions of developers. They differentiate between four groups of metrics. The first group is concerned with evaluating developers on an individual basis at a line-level, such as total amount of lines added (*creation effort*), the amount of modified lines (*modification effort*) and the amount of removed lines (*removal effort*). The second set of metrics is concerned with measuring the relationships between developers. By utilizing the effort values from the first group, it is possible to determine how many lines of added code by one developer had to be modified or refactored by another developer. With the third collection of metrics, the authors introduce concepts that indicate applied changes on a file-level. Here, Moura et al. introduce a *project revision sequence*, which makes it possible to observe which files were added, modified or deleted in the last project revision. The last group of measurements is concerned with data derived from commits, such as the total amount of commits performed by a developer [53]. The effort metrics introduced in this related work support the approach our visualization takes — highlighting creation, deletion, modification, and even movement operations in the evolution of student projects.

3.2.7 Measuring Team Members' Contributions in Software Engineering Projects using Git Driven Technology

Parizi et al. propose a solution to measure the contributions and the performance of individual students to team projects at any point of the development process. Their contribution consists of a service which automatically generates reports regarding the estimated performance of developers based on metrics such as number of created commits, amount of created pull requests, how many files were created in the project and the time spent on the project each day (see Figure 3.8). One consideration to make is that they

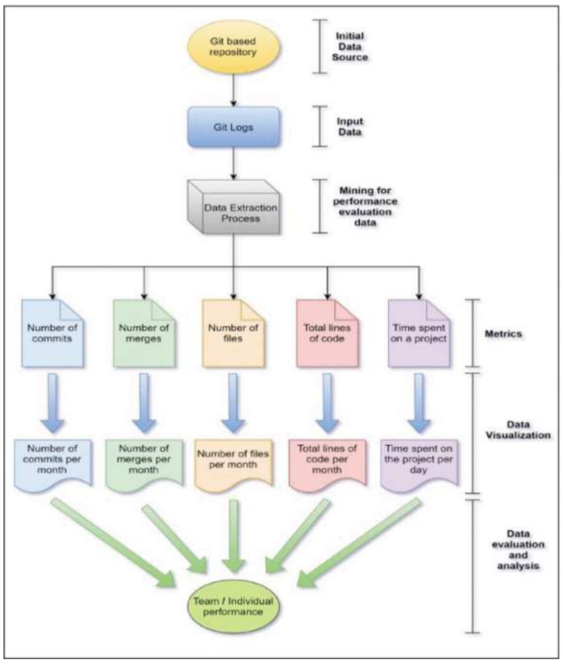

Git based repository
Initial Data Source
Git Logs
Input Data
Data Extraction Process
Mining for performance evaluation data
Number of commits
Number of merges
Number of files
Total lines of code
Time spent on a project
Metrics
Number of commits per month
Number of merges per month
Number of files per month
Total lines of code per month
Time spent on the project per day
Data Visualization
Data evaluation and analysis
Team / Individual performance

Figure 3.8: The proposed solution to measure team members contributions in software projects [60]

do not consider the scenario where a student might be working on a more sophisticated engineering problem which might lead to fewer commits while still investing the same amount of time. For this Parizi et al. suggest a difficulty gauge to include in the *weightage and performance* formula [60]. Such an additional metric to determine the actual difficulty might be something we could consider in our prototype in future iterations, in order to make the contribution analysis more precise (see Section 7.5).

3.2.8 Measure Students Contribution in Web Programming Projects

In the contribution authored by Nguyen et al., a system for assessing student group projects is proposed. Additionally, a set of metrics is presented to evaluate and analyze the contributions made to projects by individual students. Similar to the solution proposed in [53], this set is divided into individual performance and group contribution metrics keeps track of the lines of additions, modifications and deletions of code. Additionally, the number of submissions and the frequency of submissions are kept track of. Using the proposed system, Nguyen et al. found among other results that students with better grades also contributed more to their projects [55].

3.2.9 Attribution of Work in Programming Teams with Git Reporter

Guttmann et al. propose *Git Reporter*, a tool that helps teaching assistants in their grading process and the evaluation of the quality of students' contributions compared to Git. Additionally, it also helps students divide work more equally [24]. This contribution attempts to achieve something similar to our contribution, with the difference that it does not provide any form of visualization. Still, the results and ideas of this related work further inspired the approach presented in this thesis.

3.2.10 Summary & Contribution

The presented approaches provide useful solutions to visualize certain aspects of software repository evolution or assessing student contributions, but they do not combine a visualization of directory and file changes with a focus on supporting educators in grading and understanding student projects. They either do not have a project-wide view of the evolution of the structure, require checking out individual files, or are not putting the focus on the educational context.

To add to the presented research, the main contribution of this thesis was to improve the existing measurements of student projects by implementing a visualization prototype that aids student assistants and instructors in evaluating student group projects with regard to individual performance and group contribution. In the following chapter, concepts and candidates for visualization prototypes are presented and the decision of the final visualization approach detailed.

CHAPTER 4

Concepts

The following visualization concepts were designed with the software comprehension theories outlined in Section 2.4 in mind. The storyline, treemap and radial tree approaches aim to visualize the structural relations over a period of time that users otherwise would have to reconstruct mentally from commit logs. In the context of education, where tutors often quickly need to assess contributions or refactorings [48], the visualizations highlight the performed operations (e.g. adding, moving, or deleting operations) and provide information on authorship of contributions. The focus was to make the history of the structure of projects transparent and not to enable code analysis at the file-content level.

4.1 Prototype Visualizations

This section presents the prototypes of considered visualizations that were developed as part of this thesis. The three candidate mockups were evaluated to select the most appropriate visualization to display the evolution of software repositories.

4.1.1 Storyline

This visualization is inspired by an xkcd comic on movie narrative charts [88] and the publication of software evolution storylines by Ogawa and Ma [58], where certain events in a history are displayed by lines on a chart. Also, it is suggested to use real-world analogies in graphical interfaces [6] [36]. To build on these sources, the main idea was to display the structure of a software repository in a comprehensible way and to visualize the applied changes over time within this structure. Files and directories can be added, edited, moved, renamed, or deleted during this lifecycle. Each of these operations is indicated in this visualization. As the user zooms into the visualization, the level of abstraction adjusts. While Figure 4.1 and Figure 4.2 show a more abstract and compressed view of a software repository, the further a specific module is zoomed into, further files and subdirectories are shown, revealing additional details, as can be seen in Figure 4.3 and Figure 4.4. This prevents the user from being overwhelmed by too much information about the whole project.

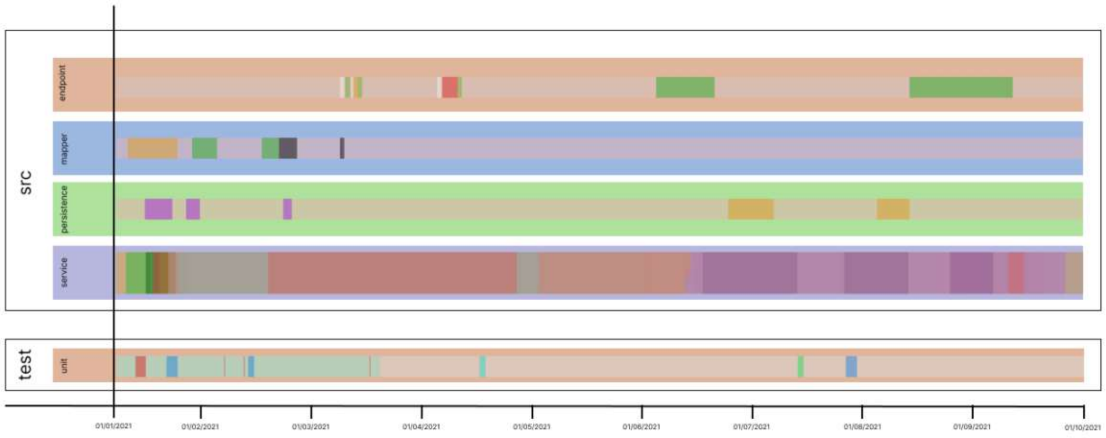


Figure 4.1: The high level view of the storyline visualization

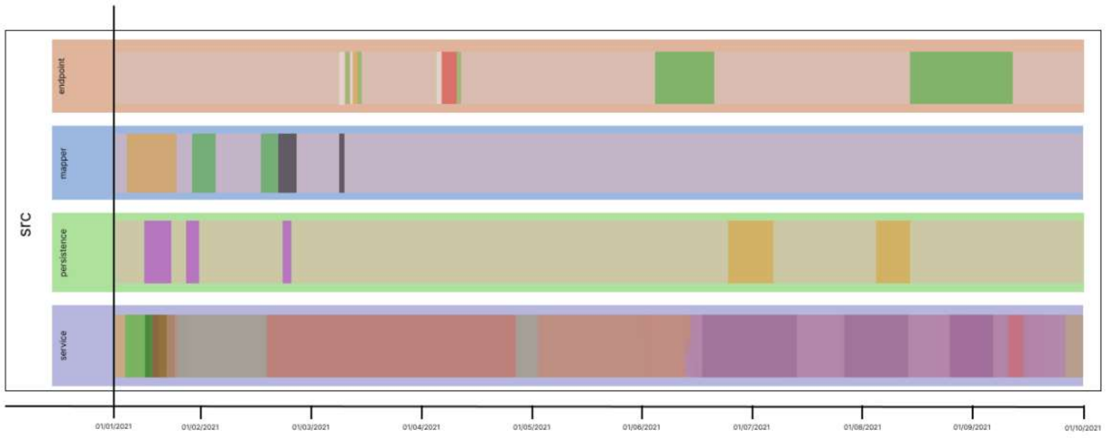


Figure 4.2: The high-level view zoomed into the *src* module

Visualizing Contributions

As it is important in an educational setting for student assistants to be able to evaluate student projects based on individual contributions, a special viewing mode was implemented. Each of the highlighted operations provides the user with information on

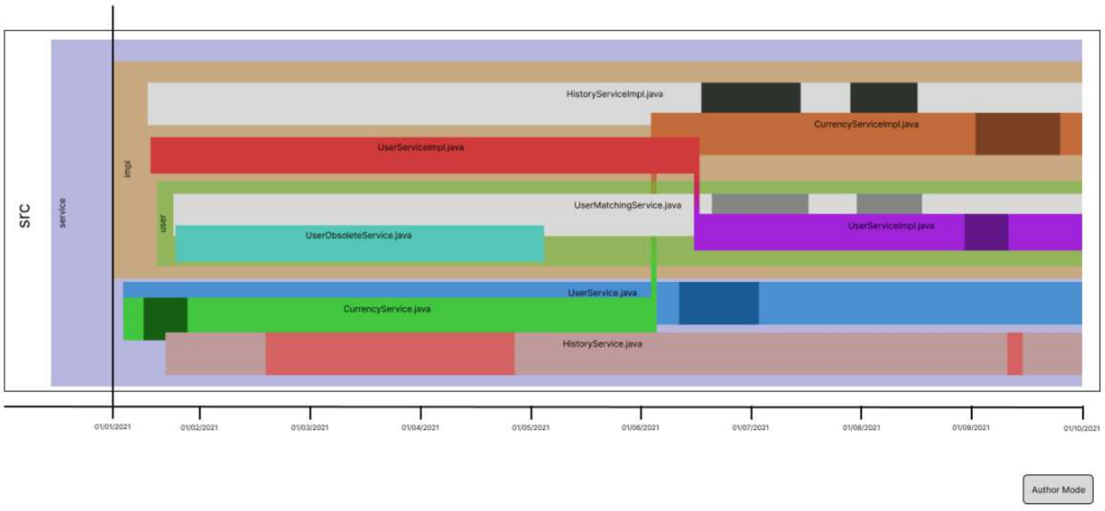


Figure 4.3: The detailed view of the *src/service* module

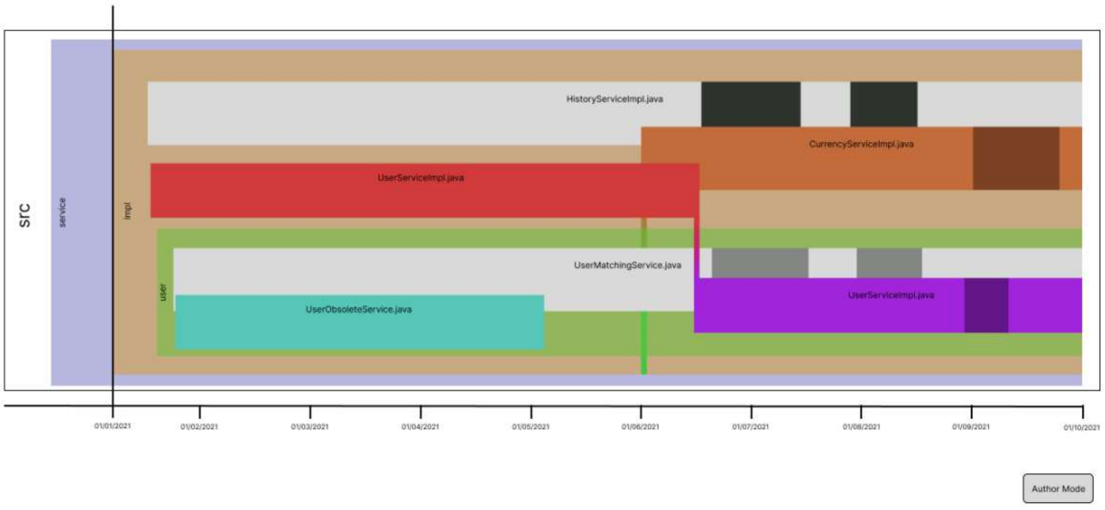


Figure 4.4: The detailed view zoomed into the *src/service/impl* module

which contributor performed it. This helps to prevent wrong judgments in the grading of student groups by making contributions of individual students more transparent.

By switching to the *author mode*, an instructor can see which student performed which operations, indicated by unique colors for each contributor (see Figure 4.5 and Figure 4.6).

Add Operation

If a file or directory is added, a new line will appear during a certain point in time. This indicates the beginning of the „history“ of this certain element. Additionally, the author

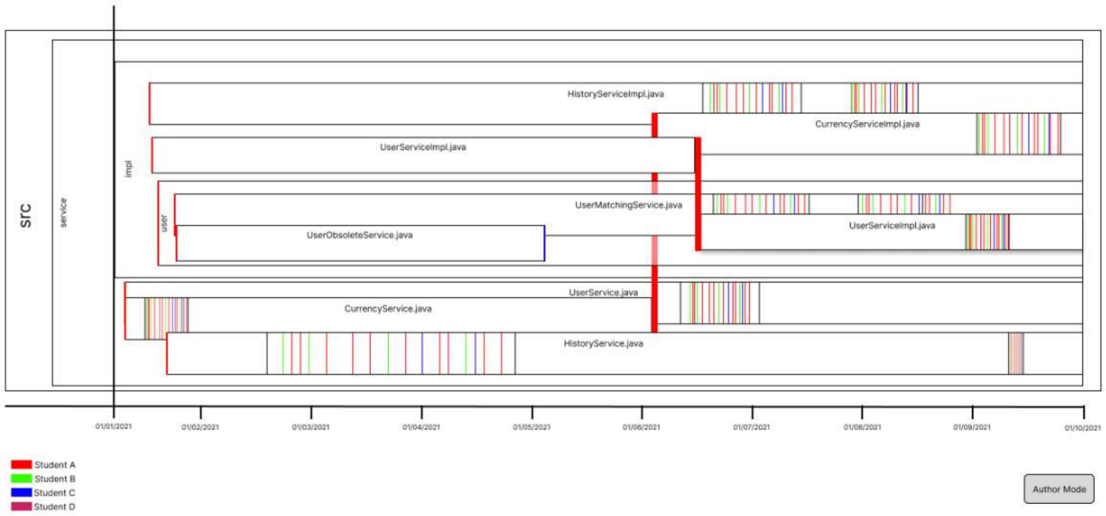


Figure 4.5: The view from Figure 4.3 in author-mode

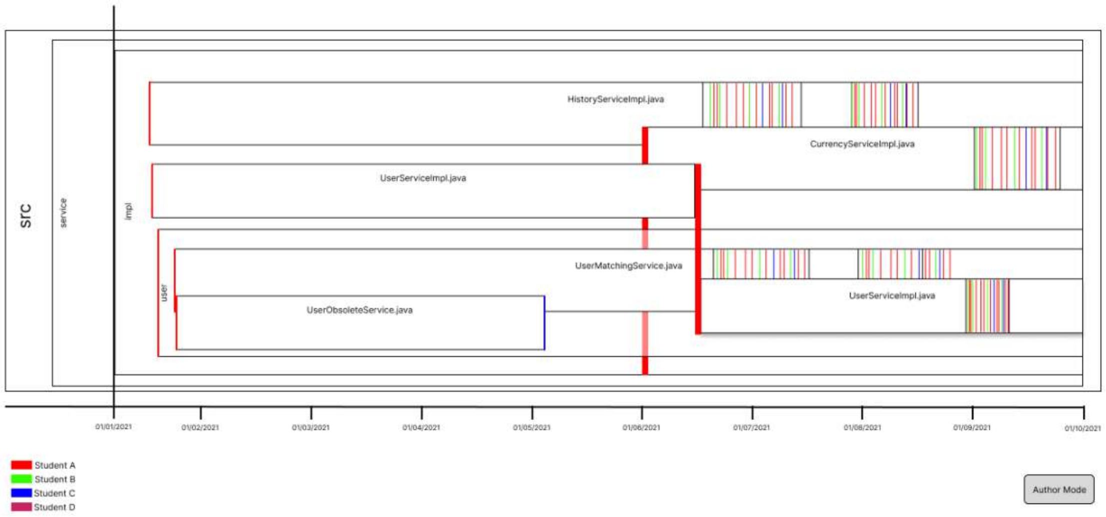


Figure 4.6: The view from Figure 4.4 in author-mode

of the file can be seen in *author mode*. This is especially useful for tutors who need to identify when and by whom files or directories were created.

Move Operation

During the lifecycle of a software project, it often occurs that files or directories are moved to different directories, e.g. during a major refactoring of parts of the project. This kind of change is visualized by a line that crosses from one directory into another one at a certain point of time. In *author mode*, instructors are able to observe the contributor that moved the file or directory. Tracking these structural changes supports the understanding of refactorings in student projects.

Edit Operation

When the contents of a file are edited, this is also shown in the *Storyline* visualization. Inspired by the visualization developed by Rozenberg et al. where changes are displayed as a series of blocks with different color saturation depending on the amount of changes that were made [66]. If the visualization is set to the *author mode*, instructors are able to see each individual commit made in a block, indicated by differently coloured lines as can be seen in Figure 4.5 and Figure 4.6. This way, tutors can detect periods of increased or decreased activity, which can help them evaluate the engagement of students over the timeline of the project.

Delete Operation

The end of the lifecycle of a file or directory is initiated by the deletion of it. Once this has occurred, the individual line on the graph ends to indicate the execution of the deletion operation. In *author mode*, the user is able to see who exactly performed the deletion.

Branch Comparison

Another feature that is contributed by this work is a visualization mode that enables the comparison of two branches that were worked on in a VCS. This can be particularly useful e.g. when attempting to comprehend a large refactoring performed by a colleague where files and directories have been moved around and renamed. Such comparisons could assist tutors or student assistants in understanding how any major refactorings or parallel development were done, without having to manually inspect details by using an IDE. As can be observed in the example portrayed in Figure 4.7 and Figure 4.8, two branches, *develop* (colored blue) and *feature/123* (colored green) are compared over a certain time-frame. Changes that are exclusive to each branch are highlighted in either blue (for *develop*) or green (for *feature/123*). If a performed operation is synchronized between both branches, it is visualized in a blue color with a green border.

Limitations

One potential limitation of this visualization is that large repositories that have many files and directories may lead to an overloaded visualization, especially with a large number of contributors.

4.1.2 Treemap

The second visualization prototype that has been developed takes a different approach to show the evolution of software repositories. It utilizes a treemap [73] to model a project's

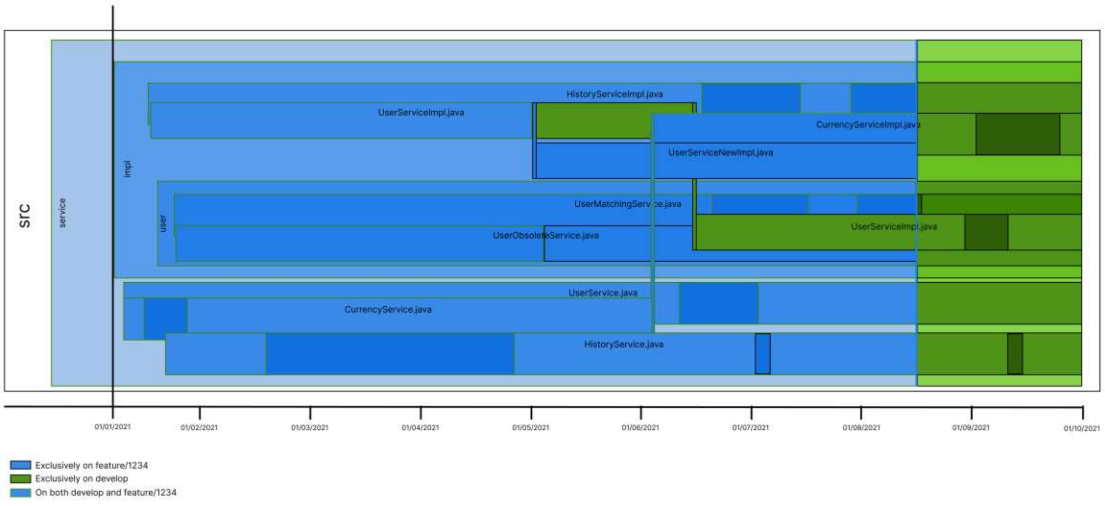


Figure 4.7: The view from Figure 4.3 while comparing two branches

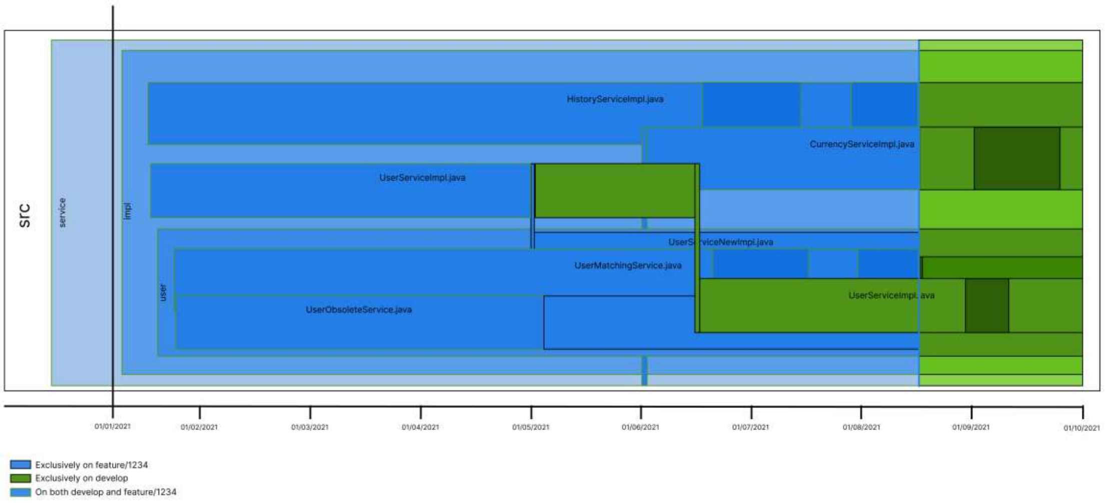
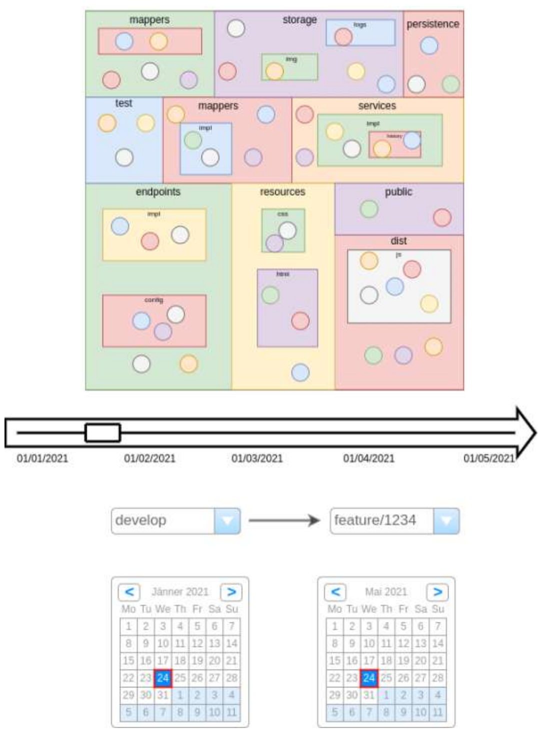


Figure 4.8: The view from Figure 4.4 while comparing two branches

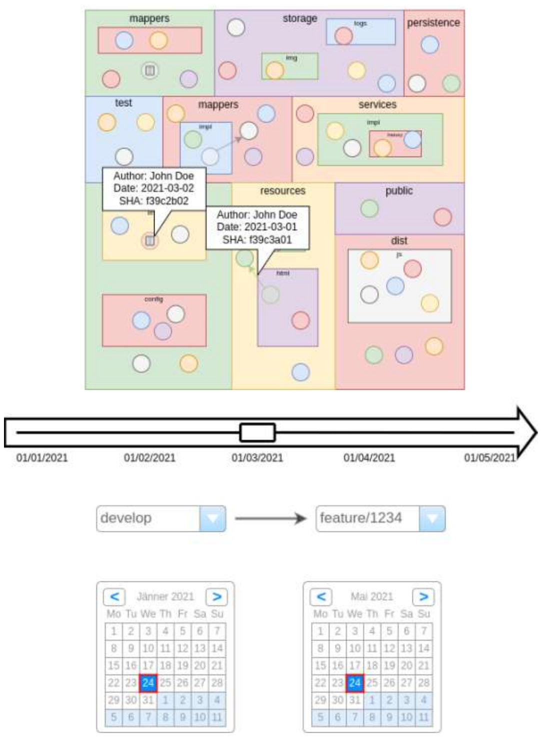
directory structure. Directories and their sub-directories are displayed as rectangles of different sizes, i.e. they are the branches of the treemap. The leaves of the treemap visualize the individual files and have the appearance of circles.

This visualization idea builds upon previous uses of treemaps for visualizing hierarchical structures [37] [47], which also applies to software repositories [19].

A major difference in the user interaction when compared to the storyline visualization presented in 4.1.1 is that the evolution over the whole timeline is not displayed in one frame. Rather, the user experiences the history of the directory tree by dragging a slider to travel through the timeline. Adjacent elements are displayed in different colors to be able to clearly differentiate them visually. Editing operations are not displayed in this kind of visualization. The add, move and delete operations for files or directories are displayed in this visualization in the following sub-sections.



(a) Base circular-treemap layout of the project hierarchy



(b) Annotated view showing how changes are highlighted

Figure 4.9: Circular-treemap mockup for visualizing a repository's directory structure.

Add Operation

When an element is added to the directory tree of a software repository, it appears at this time-point in the timeline. In the case of a directory, a new rectangle including (possibly) new files will be displayed. A file will be visualized by a new circle that appears within a directory (rectangle). This way tutors are able to see when new features were introduced by students and where the project grew during development.

Move Operation

The movements of files and directories is indicated by the change of location to a different branch of the treemap. To make this change more comprehensible, an arrow is shown in subsequent time-frames, pointing from the previous location of the element to the new one. Student instructors can keep track of refactorings more easily this way.

Delete Operation

The delete operation is visualized by the disappearance of the concerned file or directory and a trash icon that is displayed in the subsequent time-frames (see Figure 4.9b. By presenting when and where files or directories were deleted, student assistants could more efficiently understand cleanup or the removing of obsolete parts of the project by the student group.

Limitations

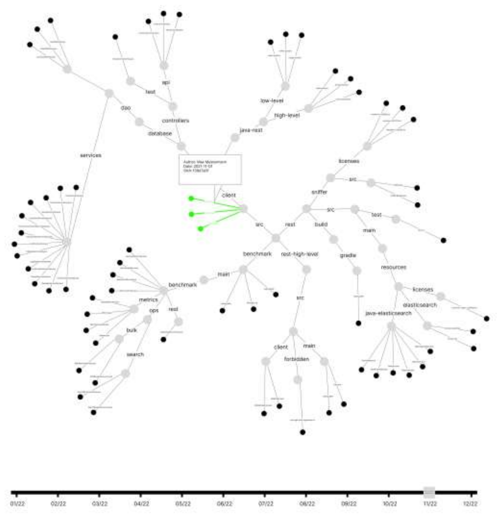
This visualization clearly visualizes the structural hierarchy of a project, but does not visualize edit operations, as it can only show one time frame at a time.

4.1.3 Radial Tree

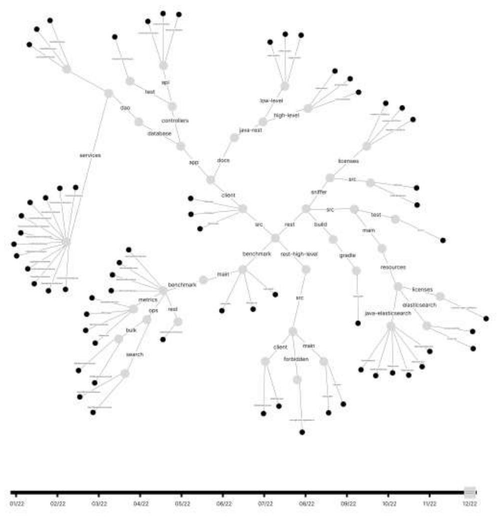
Another visualization that has been prototyped is the *Radial Tree* [73]. In this structure, folders are represented as branches of a tree. Every branch can have multiple sub-branches. The leaves of the tree display the files that are inside of directories. This representation of a repositories folder structure is able to model the following operations that occur during a certain time-frame.

Add Operation

Whenever files or directories are added during the development of a software project, this change is indicated by branches or leaves that are marked as green at that certain point in time (see Figure 4.10).



(a) Before a directory has been added.



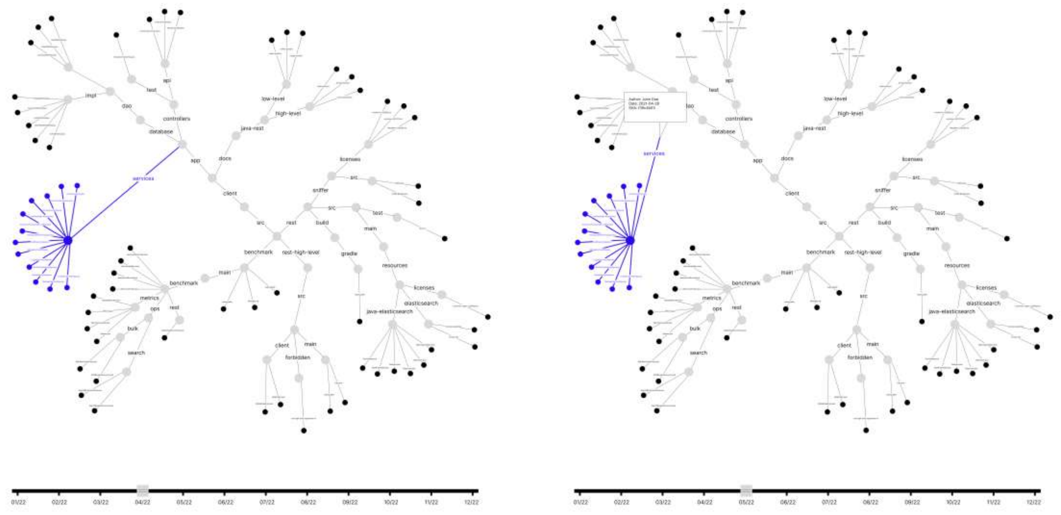
(b) After a directory has been added.

Figure 4.10: A performed add operation visualized in a radial tree.

This highlights when students extended which parts of the software projects hierarchy, making it easier to track active areas of development.

Move Operation

The movement of elements is displayed by the visualization prototype by marking the affected parts of the tree blue (Figure 4.11a) and moving them to the new location in the following time-frame, as can be seen in Figure 4.11b.



(a) Before a directory has been moved. (b) After a directory has been moved.

Figure 4.11: A performed move operation visualized in a radial tree.

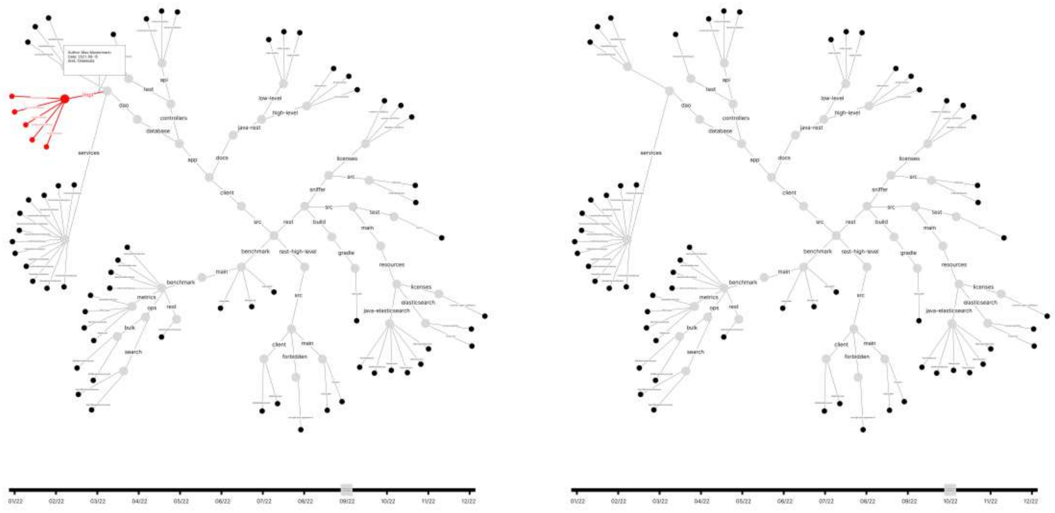
Educators can identify refactorings performed by the group, helping them understand the progress of the project they are instructing.

Delete Operation

Similar to add operations, deletions are displayed by coloring the items to be deleted in red (Figure 4.12a). In the next time-frame (Figure 4.12b), the affected nodes will be removed from the visualization. This helps tutors to see where cleanups or removal of parts of the project were performed.

Limitations

This visualization gives a good overview of the hierarchical structure, but the circular way of visualizing can make it a challenge to compare the depth across branches. Similar to the treemap, the ability to see changes over a longer period of time can be difficult.



(a) Before the delete operation. (b) After the delete operation.

Figure 4.12: A performed delete operation visualized in a radial tree.

4.2 Selection of the Prototype Candidate

In this section, we describe the process of selecting one of the developed visualizations to be used for the prototype implementation and evaluations. Each of the three candidates enables users to comprehend the evolution of software repositories with regard to adding, moving or deleting files or directories. Comparisons across different source control branches are also possible. The key difference between the visualization prototype in Subsection 4.1.1 and the methods presented in Subsection 4.1.2 and Subsection 4.1.3 is the way changes can be viewed over a certain time span. While in Subsection 4.1.1 changes over the whole timeline can be observed, the other prototypes require the user to navigate the timeline of the project in order to be able to track operations. Another important function that is offered by Subsection 4.1.1 is the ability to track movement and renaming of files (see Table 4.1). This enables users to estimate the frequency and location of changing code in order to get a better understanding of a project team's activity.

	Adding	Editing	Deleting	Moving	Branch-Comparison	Timeline
Storyline	✓	✓	✓	✓	✓	✓
Treemap	✓		✓	✓	✓	
Radial Tree	✓		✓	✓	✓	

Table 4.1: Comparison between the developed prototypes.

This aligns directly with the proposed benefits described in Section 1.2. The timeline view and the ability to track changes across it enable *Project Transparency*, which allows tutors and student assistants to see how a project evolved over time. For *Refactoring Frequency*, the ability to see file movements and renames is an important feature. Additionally, the ability to see frequently and less frequently modified areas within the project targets the *Code Hotspot & Stale Code Detection* benefit. This set of features offered by the *Storyline* visualization (Subsection 4.1.1) was the reason for the selection of this visualization concept to be developed as a prototype, as it provides the most information when compared to the other concept mock-ups.

The prototype choice was also supported by the comprehension theories discussed in Section 2.4, as the storyline visualization provides a continuous overview across an entire timeline, which reduces the working-memory load and strengthens both top-down and bottom-up comprehension [12] [75].

To validate this visualization prototype choice, domain experts such as experienced software developers were asked to complete scenarios using the prototype in an evaluation, which is presented in Chapter 6.

4.3 Mockup Evaluation

The first evaluation involved presenting the mockups described in Subsection 4.1.1 to three professionals working in the educational software development field.

The purpose of this evaluation was to find out whether the proposed benefits described in Section 1.2 can be confirmed by professionals when shown the mockup presented in Subsection 4.1.1. The evaluation provided an early validation of the chosen visualization before implementing a prototype based on the storyline mockups.

4.3.1 Evaluation Design

The evaluation was performed by interviewing the participants using a structured questionnaire. The goal was to assess the perceived usefulness of the proposed visualization technique and if they would use it in their professional practice. Additionally, questions about potential improvements were asked. Each participant was asked the same questions and shown the same mockups in a consistent order to ensure uniformity in the evaluation process.

4.3.2 Evaluation Results

The feedback on the initial mockup was overall very positive. The visualization was found to be useful for various educational applications, especially for gaining an overview of student contributions and the evaluation of a software repository. Several concrete suggestions for improvements were made, especially with regard to coloring, line thickness, clarity of the branch comparison feature, and the visualization of overlapping elements.

Profession

The participants were split among two professional roles, with the majority being software developers (see Figure 4.13). Their experience varied, with most professionals having 0-3

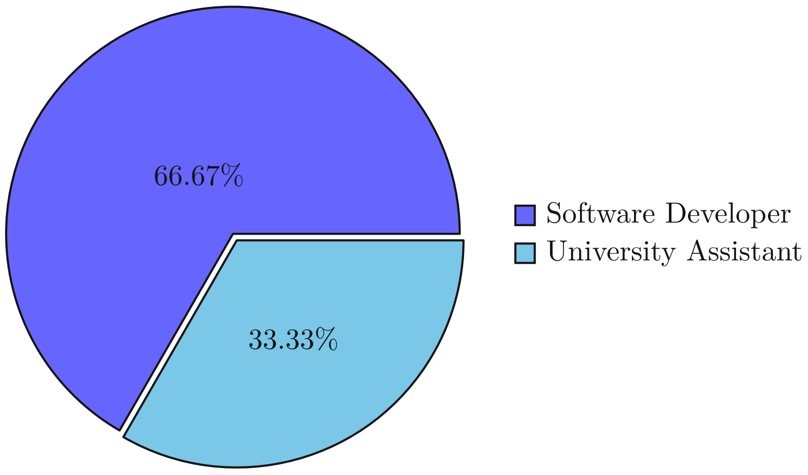


Figure 4.13: Roles of Participants

years of experience and one 5-7 years, as can be seen in Figure 4.14. This indicates that the feedback is mainly from less experienced professionals, which may have an influence on the perceived usefulness of the presented tool.

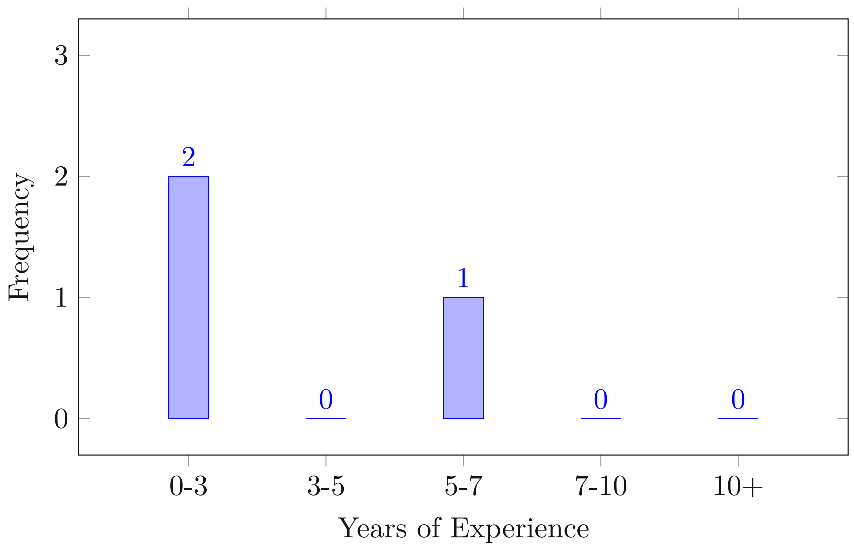


Figure 4.14: Years of Professional Experience of Participants

Visualizing Operations

The usefulness of the visualization of individual file operations such as creating, moving, or deleting was perceived as very good, with all participants rating it with a clarity of 5. The results are displayed in Figure 4.15. Some suggestions were made including making the connection lines thicker and reducing overlapping of storylines. This suggests that this part of the visualization tool is effective in showing these operations in a clear way, which can be crucial for understanding the history of a software project.

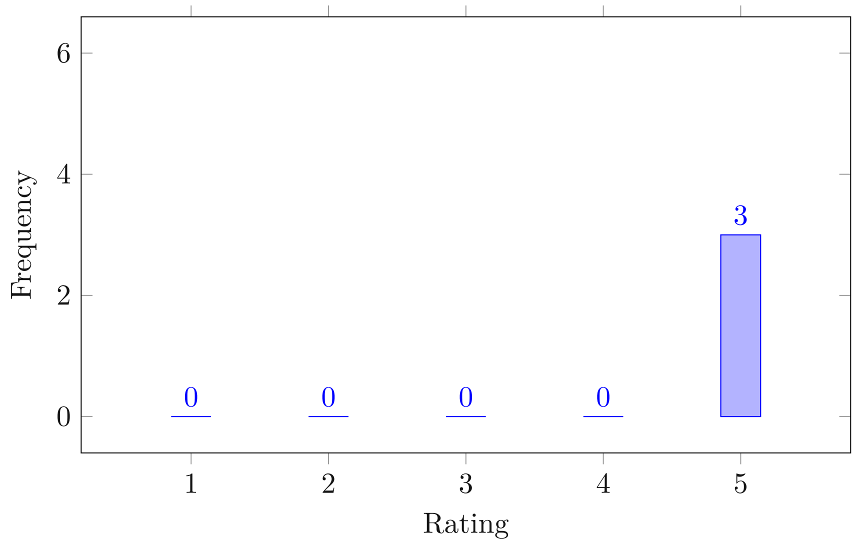


Figure 4.15: How clearly are the individual operations (create, move, delete) visualized?

Code Frequency Analysis

Regarding the overall usefulness of the tool to visualize code change frequency in software repositories, the participants unanimously rated with a scoring of 4 (see Figure 4.16). Being able to detect time frames of inactivity (potential dead code) was received positively, but the participants also mentioned that they saw limitations in gauging the level of activity. There were suggestions to make the colors darker if more changes were made in that specific time frame and vice versa. Also, clearer time markers such as vertical lines were identified as potential improvements. This may indicate that there is room for improvement in order to achieve a higher level of satisfaction.

Contribution Analysis

Visualizing individual contributions with the *author mode* was received positively with the majority assigning a scoring of 5, while one participant rated it a 4. It was considered helpful especially for tutors or student assistants to identify the student contributions within a group. There was also the feedback that the colors should be clearer and better

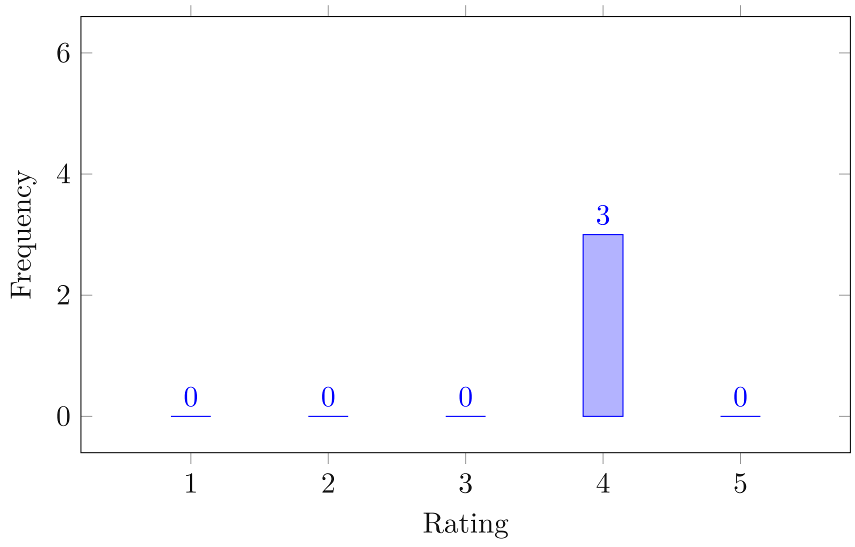


Figure 4.16: Usefulness of the visualization to determine code change frequency.

distinguishable while being higher in contrast. The thickness of the lines was seen as not uniform, and there was the wish for filtering options when many contributors are displayed. Another suggestion was to visualize contribution shares proportionally within the storylines in order to reduce bias from visually more dominant colors. Overall, the feature was described as very valuable for educational use cases.

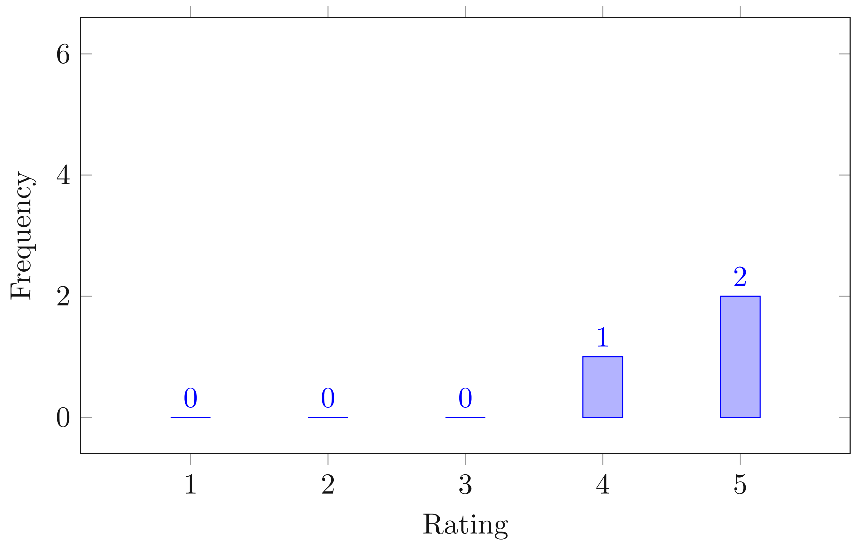


Figure 4.17: Usefulness of the visualization to determine individual contributions.

Branch Comparison

The third feature of the visualization mockups was concerned with visualizing changes among different branches. The feedback on this part of the tool was mixed but overall still positive, scoring mainly ratings of 4 and one rating of 5. See Figure 4.18 for the visualized results.

The ability to track differences between branches was found to be useful, while the color contrast was seen as sometimes not enough, with the distinction between „blue with border“ and „only blue“ not always being clear. Improvement suggestions included considering a higher contrast, more color options, and customization of coloring, especially when considering accessibility. One of the participants also commented that it should be more clearly indicated that a file continues to exist even after merging.

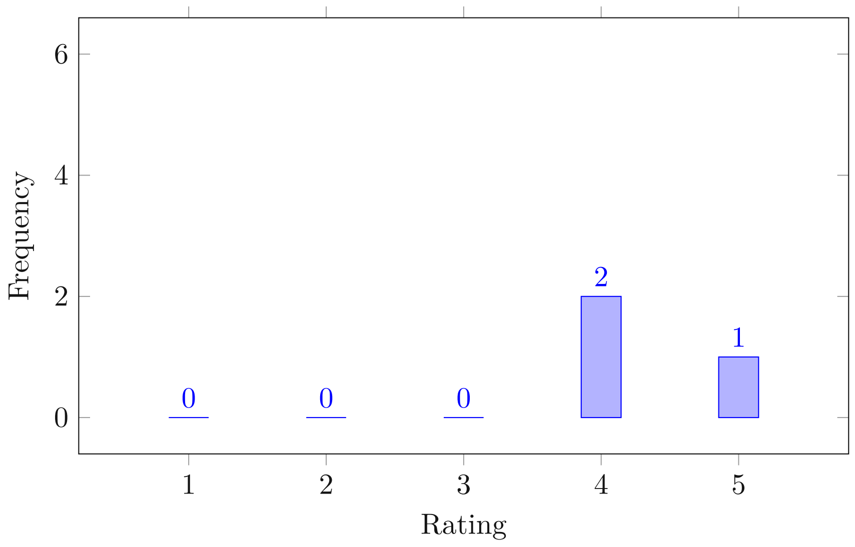


Figure 4.18: Usefulness of the visualization regarding branch comparison.

The participants all rated the use case of analyzing refactorings made by contributors with a scoring of 5 (see Figure 4.19), suggesting that this would be a common task in the daily life of the professionals.

Overall Prototype

Overall, the visualization mockup was received positively, with one participant rating it as 4 and the others assigning a score of 5, as can be seen in Figure 4.20. The visualization was seen as useful for providing a clear overview of project structure and contributions. Additionally, it was emphasized that filters and customization was needed to prevent having too much information displayed at the same time, especially in large projects. The implementation of the mockups should also consider file size or importance, to add even more context.

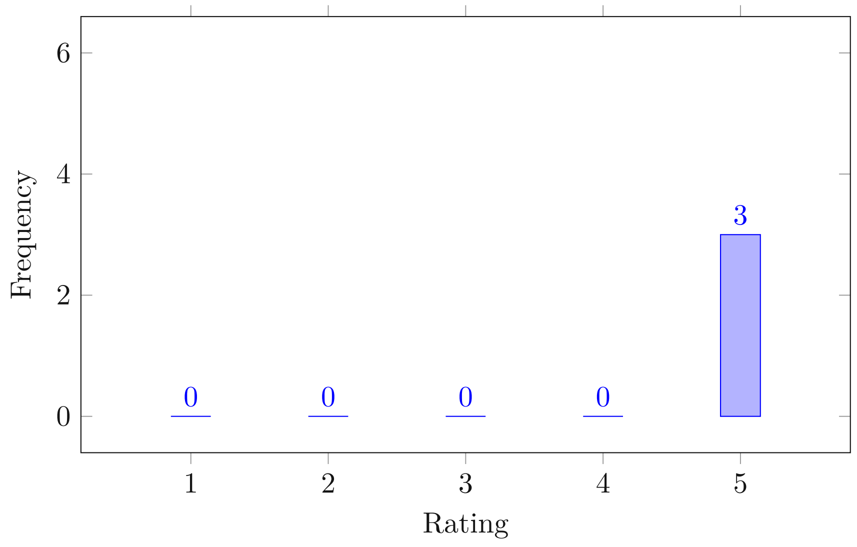


Figure 4.19: Usefulness of the visualization for analyzing refactoring tasks.

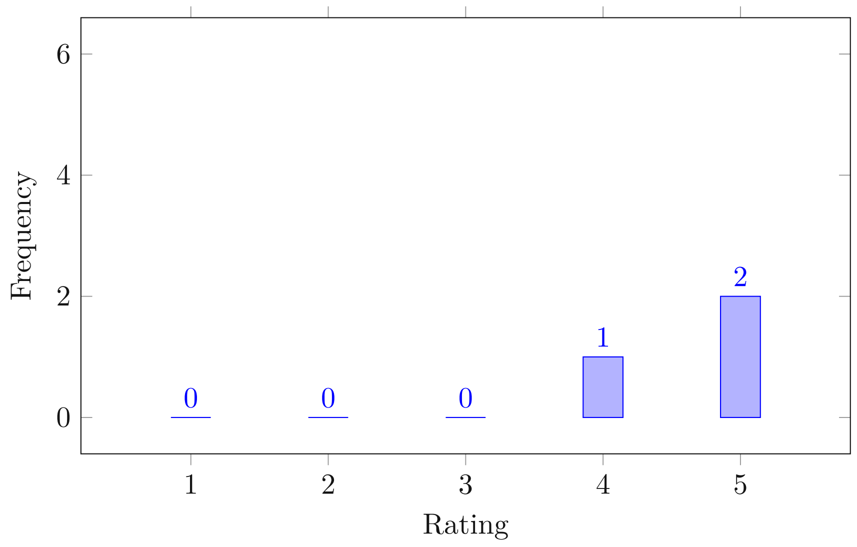


Figure 4.20: General usefulness of the visualization prototype

These results confirmed that the storyline mockup chosen in Chapter 4 addressed the targeted benefits (see Section 1.2).

Based on the feedback received during this first evaluation round, the scenarios and tasks for the subsequent prototype evaluation were created (see Subsection 6.1.5). Each of the scenarios was designed to test features that had been highlighted as useful or that needed to be improved. The folder analysis scenario targeted the visualization of operations, the student contribution scenario focused on the author mode. Refactor analysis tested how well renaming and moving of files could be detected, and the code frequency scenario evaluated the clarity of change frequency over time. This way, it was ensured that the evaluation scenarios and tasks covered use cases in software engineering education, such as tutors preparing for meetings with their student groups, and also reflecting the feedback given during this mockup evaluation.

4.3.3 Summary and Educational Implications

The visualization concepts presented in this chapter directly address the first research question, **RQ1**, which asked which different approaches to visualize the directory structure history of a software project exist. These three visualizations were compared based on their ability to represent important operations such as adding, moving, editing, deleting and comparing branches and their ability to provide an overview spanning a certain time-frame, which is summarized in Table 4.1. **RQ2** was then addressed in the mockup evaluation presented in this Section by performing the described evaluation.

Some limitations became evident in these visualizations during the mockup creations and their comparison. Treemaps and radial trees are only able to display one point of a timeline at once. This is in contrast to time-oriented visualizations. Nawrot and Doucet state that „It is established in scientific literature that time-oriented data visualizations can increase information retrieval performance, aid memorization and insight discovery processes“ [54].

An in-depth comparison across branches could also be difficult with the usage of Radial Trees. The Storyline visualization could make it easier to comprehend evolution over time [54], but might not scale very well in large projects with a lot of contributors and operations.

From an educational perspective, each visualization seems to have strengths. With Treemaps, the structural hierarchy could help to understand how a student project grew or shrank over time [12]. Radial Trees could be useful to understand which branches of a project were most actively developed. But the Storyline visualization gives a time-oriented overview of the student project and also links to individual contributors. As also stated above, the possible increased insight discovery processes [54] could make it easier to grade in a fair way and assess projects more quickly for educators.

It is important to note here that while the branch-comparison mode was conceptually evaluated here, it was omitted in the prototype implementation itself, as described in Section 5.1.

CHAPTER 5

Prototype Implementation

Based on the feedback gathered during the mockup evaluation (see Section 4.3), a prototype was developed using a modern web-technology stack. The goal of the implementation was to provide a way for the participants of the prototype evaluation that followed (see Section 6.1) to interactively explore the storyline visualization. The implemented prototype consisted of three modules, a frontend, a backend and a shared library used by both.

With the boundaries described in Section 5.1 clarified, the subsequent sections describe the implementation details of the final visualization prototype and present its features.

5.1 Scope Limitations of the Prototype

During the development of the prototype, it became evident that the implementation of the branch-comparison mode would take too much time to be within the scope of this thesis. It was decided to omit the feature and focus on the remaining parts of the prototype. This is both reflected in this chapter and in Section 6.1, where this specific functionality is not taken into consideration.

Additionally, the implemented prototype has some minor visual differences from the mockup visualizations. These differences were caused by the limitations of the technologies used in the prototype and feedback received during the initial evaluation round. Despite these visual changes, the core functionalities are preserved, with the exception of the branch-comparison mode, which was omitted.

5.2 Data Structure

The frontend module (see Section 5.4) requires information in a specific format to render the visualization prototype. This semi-structured JSON data is generated by the backend after completing the mining process described in Section 5.3. The data structure is designed as a tree-like hierarchy containing a list of objects, which can be either `Directory` or `File`. To refer to either a `Directory` or a `File`, we use `FileSystemItem` from here on.

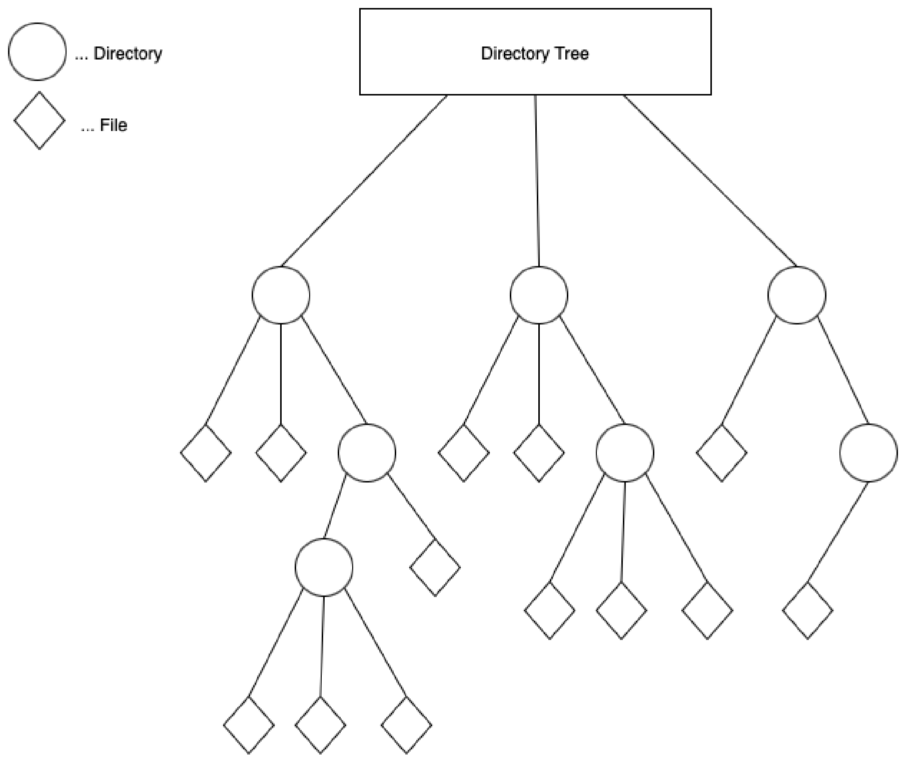


Figure 5.1: The hierarchical structure of the directory tree, created using draw.io.

Directories can contain one or more `FileSystemItem` in their `contents` field, which recursively nests to describe the hierarchy of the tree-like structure, the *directory tree*, as illustrated in Figure 5.1.

Each `FileSystemItem` has a `filepath` to describe its location and a `changeElements` field to track changes made over time. This field is an array of objects that describe changes made to the `FileSytemItem` over time, which we will refer to as `ChangeElements` (also see Figure 5.2). Each `ChangeElement` includes the following attributes:

- **status**: Indicates the type of change (`added`, `modified`, `removed`, `moved`).
- **timestamp**: Records when the change occurred.
- **commit**: Contains information about the commit that made the change, such as the commit message and the commit author. This commit object includes:
  - **oid**: The object ID of the commit.

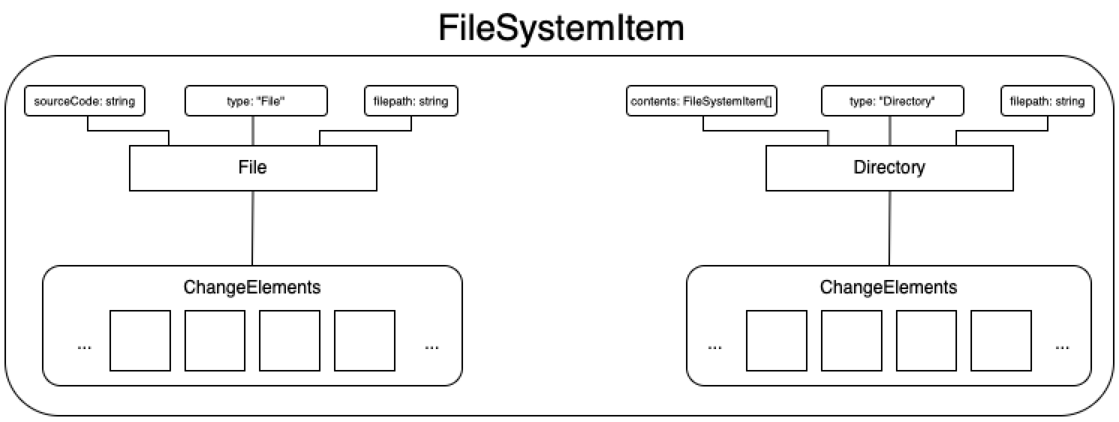


Figure 5.2: The structure of a `FileSystemItem`, created using draw.io.

- **message**: The commit message.
- **parent**: An array of parent commit IDs.
- **tree**: The tree object ID associated with the commit.
- **author**: Details about the author of the commit (name, email, and timestamp).
- **committer**: Details about the committer of the commit (name, email, and timestamp).
- **branch**: The branch name associated with the commit.

- **filepath**: The path of the `FileSystemItem` the `ChangeElement` belongs to.
- **successor**: The successor of the `ChangeElement`, only set if is involved in a movement operation.

5.3 Backend

The backend module operates in a Node.js environment and was implemented using TypeScript [86], isomorphic-git [83] for processing Git repository data and tree-sitter [82] for generating Abstract Syntax Trees.

In order to convert given software repositories to the data structure described in Section 5.2, which is then consumed by the frontend module, the backend module performs several operations on the fetched repository data.

The software repository is first cloned from a platform such as GitHub or GitLab by specifying the repository URL, e.g. `https://github.com/YOUR-USERNAME/YOUR-REPOSITORY` (see `getHistory` in the source code). Then, each commit that was made over the history of that project is processed in the pipeline described in the following subsections, which is also visualized in Figure 5.3. After the processing is finished, a directory-tree containing data about the performed operations, including `FileSystemItem` movement data, is emitted by the backend in the form of a JSON file, to be consumed by the frontend module.

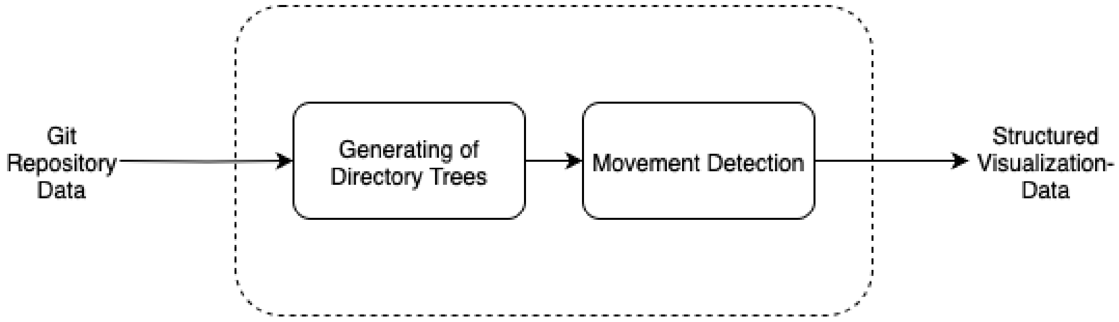


Figure 5.3: The repository mining data pipeline, creating using draw.io

5.3.1 Generating of Directory Trees

The data of the cloned repository is then processed in the first phase (also in `getHistory` in the source code). Each fetched commit is analyzed by walking through its file tree and comparing it to its parent commit.

This way, it is determined whether the `FileSystemItem` was `added`, `removed` or `modified`. The source code of the `FileSystemItem` is also extracted. The `FileSystemItem` is then added to the directory-tree together with an initial `ChangeElement` entry. If the `FileSystemItem` already existed in the directory-tree, another `ChangeElement` is added to the existing item.

The `ChangeElement` entries contain the determined `status` and other data. The source code extracted earlier is stored with the `FileSystemItem` in the directory-tree. The data structures of `ChangeElement` and `FileSystemItem` are defined in more detail in Section 5.2. See `insertIntoDirectoryTree` in the backend module to inspect the source code for this part of the processing pipeline.

5.3.2 Movement Detection

Once the directory-tree was successfully created, the second processing phase begins. During this phase, the tree is processed by two sub-algorithms. First, it is analyzed to detect whether any files or directories have been moved. A heuristic is used to perform this detection. Each `ChangeElement` has a `status` field that indicates which operation was performed, which is modified by the second sub-algorithm in case of a detected movement operation.

Grouping of **`ChangeElements`** across **`FileSystemItems`**

The first sub-algorithm, referred to as `groupChangeElementsByKey` in the source code, groups `ChangeElements` with the status `removed` or `added` by their timestamps.

Additionally, the heuristic uses the file name of the corresponding `FileSystemItem`, which is taken from its `filepath`. A `SHA-256` hash and an AST of its source code are then created to be used in the following steps.

The heuristic attempts to detect moved `FileSystemItems` by grouping any of the `ChangeElements` in the directory-tree (referred to as `fileHistory` in the source code) across all items that either have the status `removed` or `added`. There are two ways that these elements are then grouped, which are implemented in `groupChangeElementsByKey`.

One grouping is done by their contents. Based on the source code of each `File` that the `ChangeElements` under comparison belong to, the Abstract Syntax Trees (AST) is calculated. The prototype implementation only supports this for `.ts`, `.tsx` and `.java` files. The *Tree-sitter* parsing library [82] was used to achieve this. If the weighted similarity of the ASTs of two compared source codes is greater than or equal to 50%, they are considered to be similar and the `ChangeElements` are grouped together. The logic regarding AST is implemented in `movedFileDetectionService.ts` and `ast.ts`. These grouped elements are then stored in a hash map under the `SHA-256` hash of the contents.

The second grouping is done in a similar way, but by the names of the compared `Files`. So if there is a `File` that was deleted at one point in time and at the same time, another `File` with the same name shows up, they will be grouped together in another hash map under the same name. The steps described above are visualized in Figure 5.4.

Transforming to Moved `ChangeElement`

In the second sub-algorithm, referred to as `transformDeletedAndAddedToMoved` in the source code, the two hash maps are then sorted by their timestamp keys. Each entry of a hash map contains `ChangeElements` that are considered as being involved in a movement operation.

For each of these entries, the `ChangeElement` with the status `added` is considered as the movement destination (referred to as `newChangeElement` in the source code). The other elements in the same group are then assigned the path of `newChangeElement` as `successor` and their status to `moved` (see Figure 5.5). This enables the frontend later on to link these files together visually (see Section 5.4).

5.3.3 Considerations

This algorithmic approach was inspired by the way Git performs detection of file similarity in combination of utilizing AST, as presented by Fujimoto et al. [21], while being kept within the scope of this thesis. There are limitations to this approach, false positives and negatives are to be expected and were encountered during implementation and had to be manually adjusted in the dataset used for the prototype evaluation. The heuristic was considered a best-effort approach to explore the idea and provide the visualization part

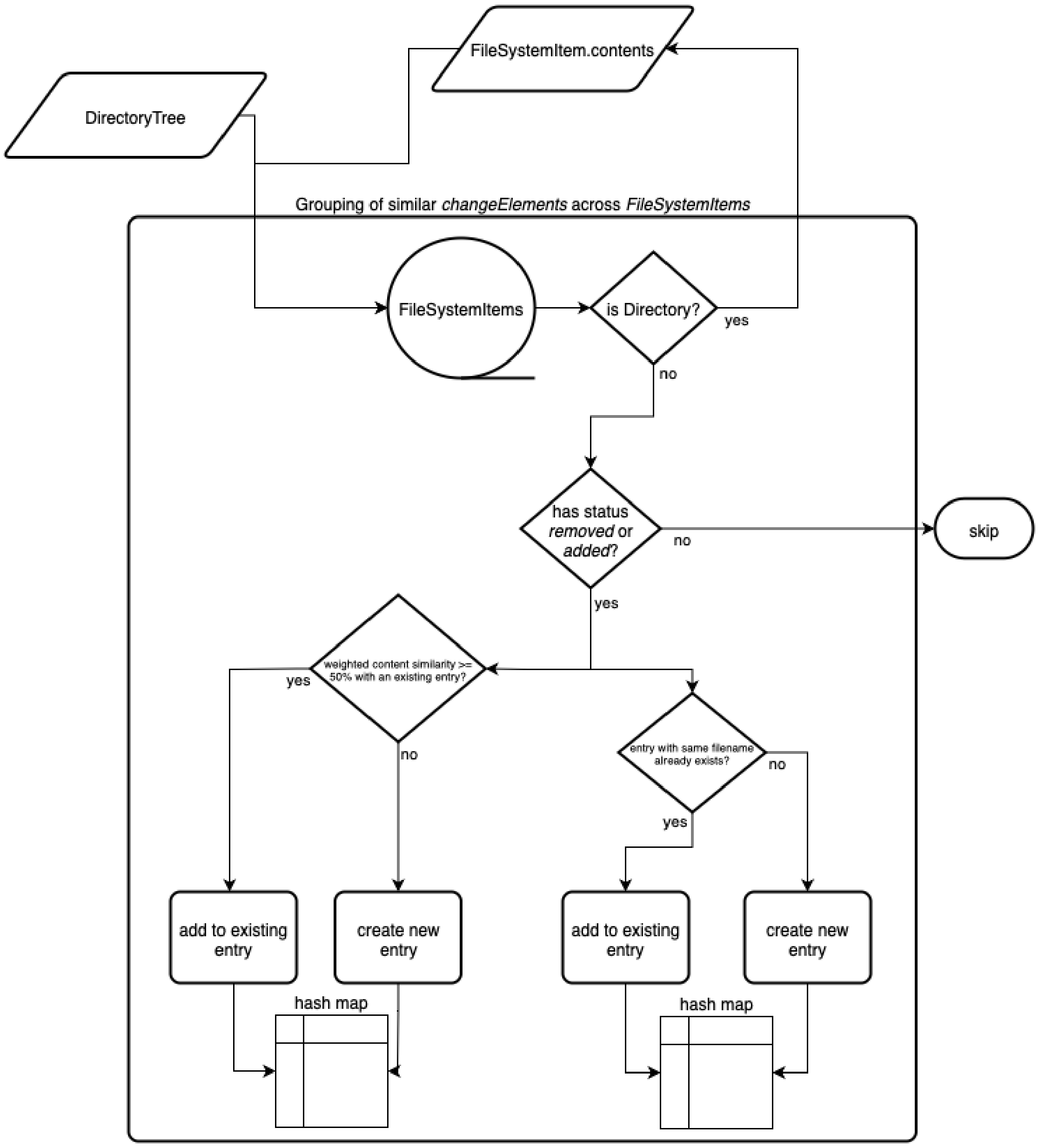

FileSystemItem.contents
DirectoryTree
Grouping of similar *changeElements* across *FileSystemItems*
FileSystemItems
is Directory?
yes
no
has status *removed* or *added*?
no
skip
yes
weighted content similarity >= 50% with an existing entry?
yes
no
entry with same filename already exists?
no
yes
add to existing entry
create new entry
add to existing entry
create new entry
hash map
hash map

Figure 5.4: The algorithm for grouping `ChangeElements` across `FileSystemItems` to detect movement operations.

of the prototype, the frontend, with a semi-structured dataset derived from a software repository.

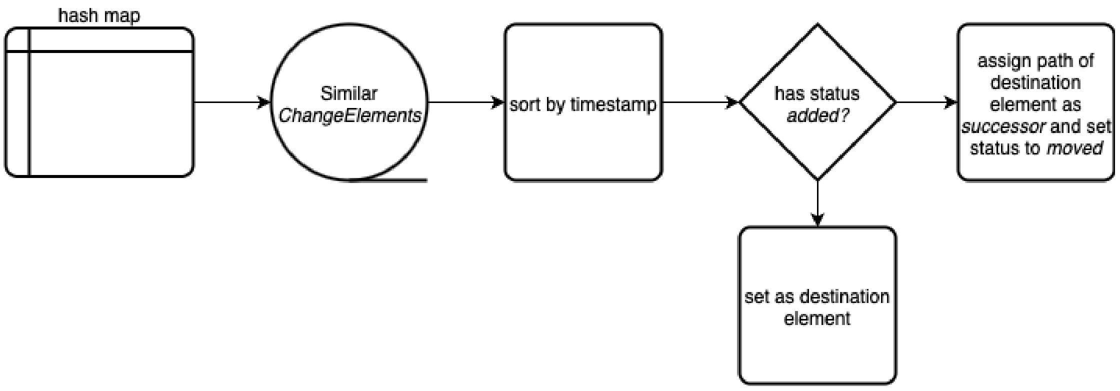

hash map
Similar *ChangeElements*
sort by timestamp
has status *added?*
assign path of destination element as *successor* and set status to *moved*
set as destination element

Figure 5.5: The algorithm for transforming similar `ChangeElements` determined in the previous step to the `moved` status.

5.4 Frontend

The frontend module implements the storyline visualization (see Subsection 4.1.1). Given a payload of the structure described in Section 5.2, a graph is rendered in the browser that implements the mockups of the proposed visualization.

The prototype offers the same set of features as presented in the mockups (see Subsection 4.1.1), with the exception of the branch-comparison mode, as explained in Section 5.1.

5.4.1 General

In the default mode, the user starts at the top-level view of the directory tree, as shown in Figure 5.6, and can navigate further down in the file hierarchy, revealing further files and directories (see Figure 5.7). To go back up one level, the „Go one level up“ button can be used. The current path is always displayed underneath the visualization, ensuring that users can always see their location within the directory tree during navigation.

There are two additional buttons below the path information and next to „Go one level up“. The *Author Mode* and *Contribution Mode* buttons enable the user to switch between these modes. Another click deactivates the respective modes again. These UI elements are shown in Figure 5.8.

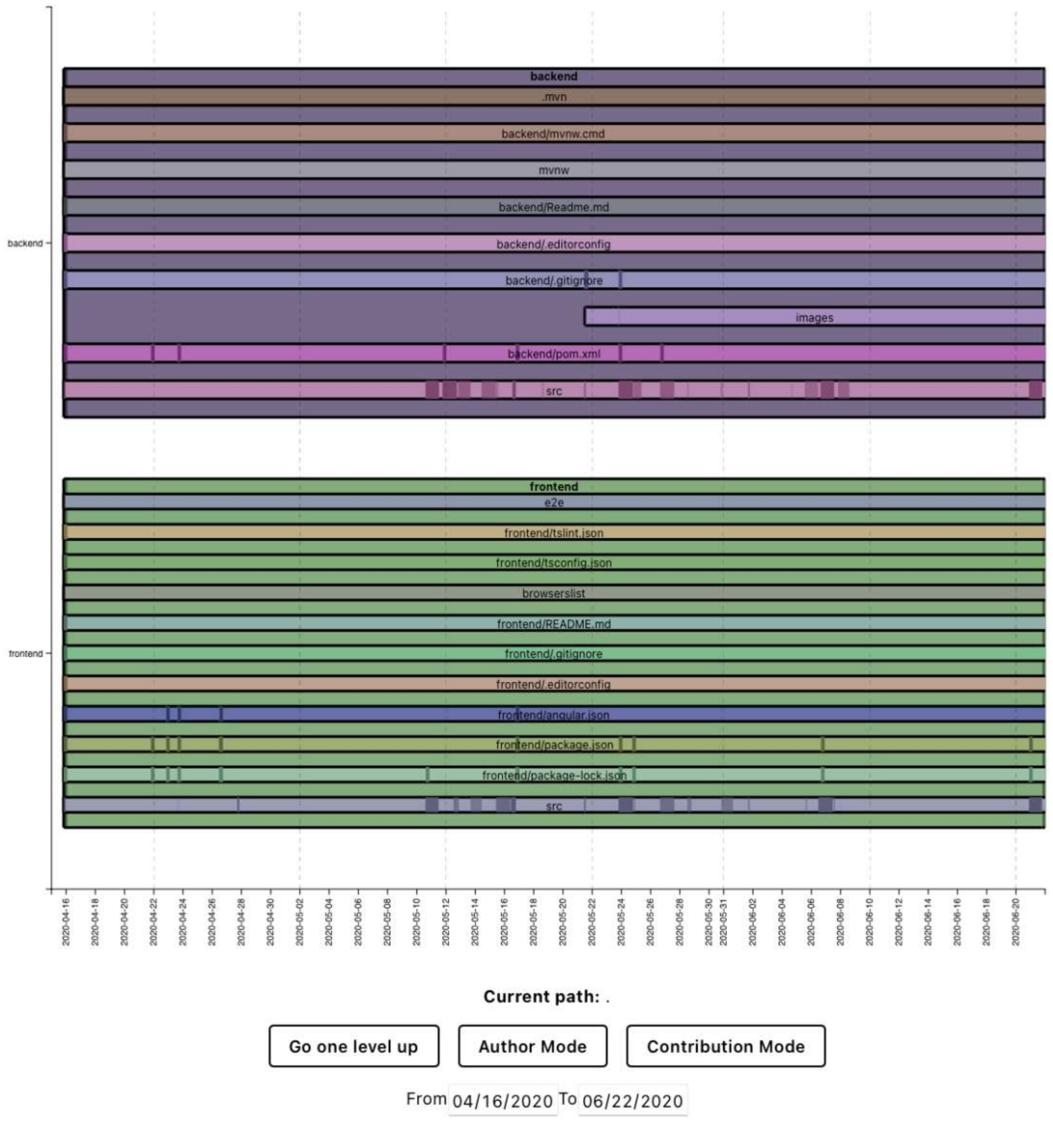
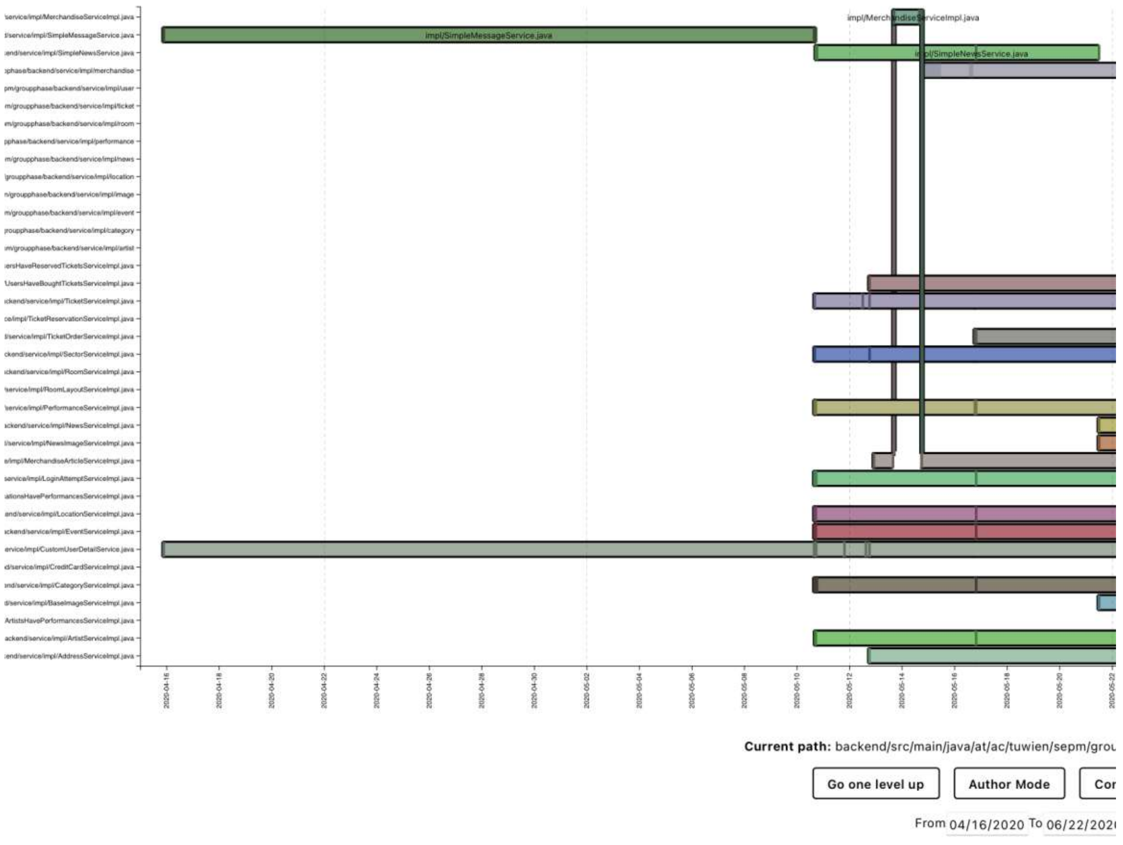


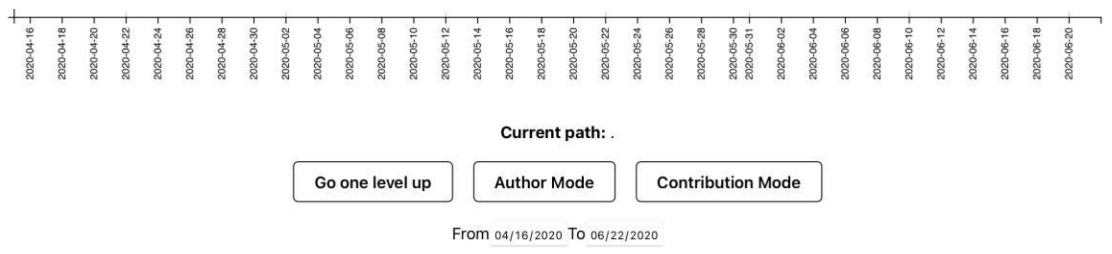
Figure 5.6: A top-level view in the default mode of the implemented prototype.


impl/SimpleMessageService.java
impl/MerchandiseServiceImpl.java
impl/SimpleNewsService.java
Go one level up
Author Mode
Cor

Current path: backend/src/main/java/at/ac/tuwien/sepm/grou

From 04/16/2020 To 06/22/202

Figure 5.7: A more detailed view in the default mode of the implemented prototype.



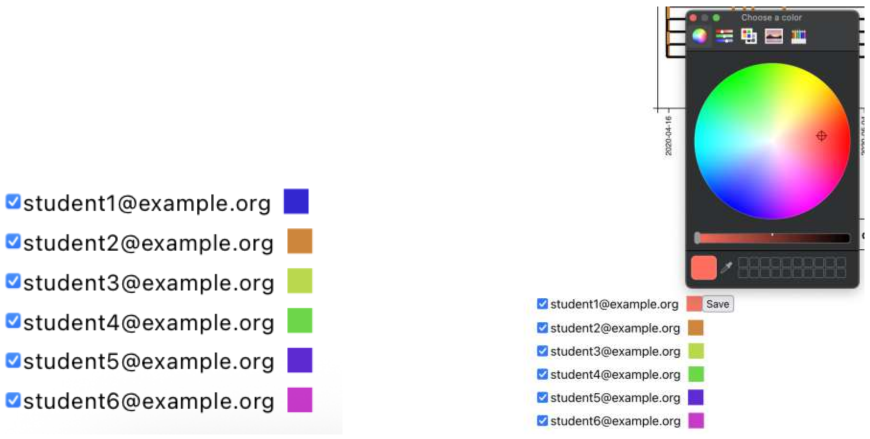
Current path: .

From 04/16/2020 To 06/22/2020

Figure 5.8: The UI elements of the prototype that enable switching between modes and navigating levels.

5.4.2 Author Mode

Any view can also be displayed in *Author Mode*, which allows the user to see which changes were made by which contributor (see Figure 5.10 and Figure 5.11). The *Author Mode* displays a list of contributors, each assigned a unique color, underneath the UI elements (see Figure 5.9a).



student1@example.org
student2@example.org
student3@example.org
student4@example.org
student5@example.org
student6@example.org

(a) The list of contributors, each assigned a color.

(b) Colors can be overridden by selecting a custom color from a palette.

Figure 5.9: The contributor list displayed in *Author Mode* and *Contribution Mode*.

The colors are unique as they are assigned by creating a hash of the contributor email and name. These colors are also used in the storylines themselves, replacing the colors that are used in the default mode.

Only contributors with visible changes in the current view are listed. The list of contributors can be filtered by hiding specific individuals, which is immediately reflected in the visualization. Additionally, users can change the assigned colors to accommodate visual impairments, such as red-green color blindness, ensuring better visibility of changes (see Figure 5.9b).

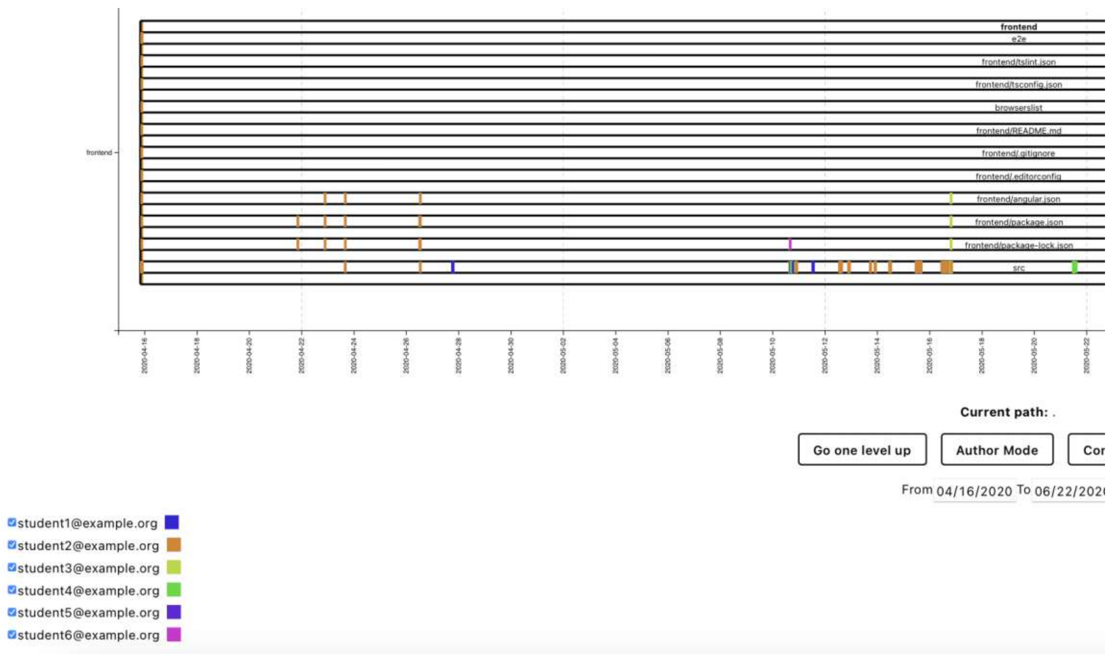
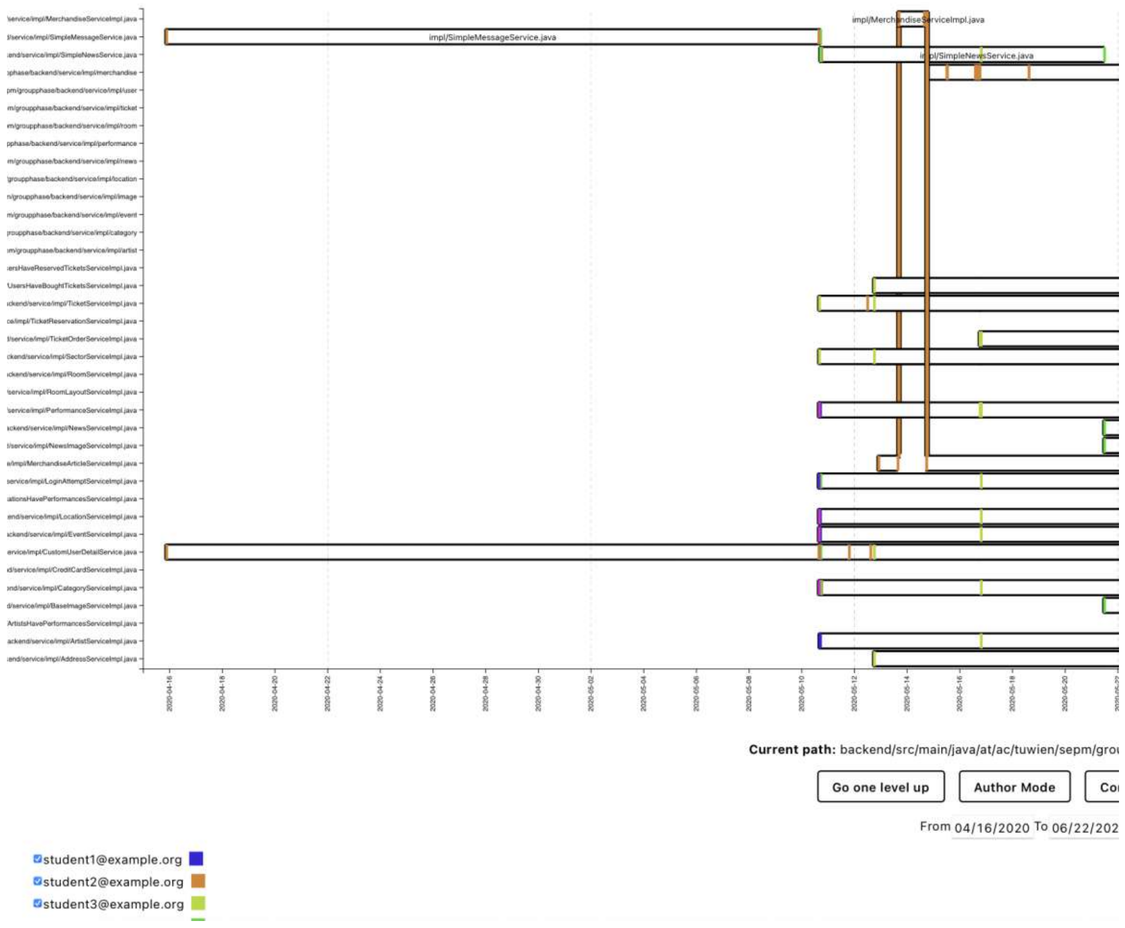


Figure 5.10: *Author Mode* in a top-level view.


'service/impl/MerchandiseServiceImpl.java
l/service/impl/SimpleMessageService.java
:end/service/impl/SimpleNewsService.java
:phase/backend/service/impl/merchandise
pm/groupphase/backend/service/impl/user
m/groupphase/backend/service/impl/ticket
m/groupphase/backend/service/impl/room
pphase/backend/service/impl/performance
m/groupphase/backend/service/impl/news
'groupphase/backend/service/impl/location
n/groupphase/backend/service/impl/image
m/groupphase/backend/service/impl/event
roupphase/backend/service/impl/category
:m/groupphase/backend/service/impl/artist
:ersHaveReservedTicketsServiceImpl.java
UsersHaveBoughtTicketsServiceImpl.java
ickend/service/impl/TicketServiceImpl.java
ce/impl/TicketReservationServiceImpl.java
l/service/impl/TicketOrderServiceImpl.java
ckend/service/impl/SectorServiceImpl.java
ickend/service/impl/RoomServiceImpl.java
'service/impl/RoomLayoutServiceImpl.java
'service/impl/PerformanceServiceImpl.java
ackend/service/impl/NewsServiceImpl.java
l/service/impl/NewsImageServiceImpl.java
e/impl/MerchandiseArticleServiceImpl.java
service/impl/LoginAttemptServiceImpl.java
ationsHavePerformancesServiceImpl.java
:end/service/impl/LocationServiceImpl.java
ackend/service/impl/EventServiceImpl.java
ervice/impl/CustomUserDetailService.java
d/service/impl/CreditCardServiceImpl.java
ond/service/impl/CategoryServiceImpl.java
d/service/impl/BaseImageServiceImpl.java
ArtistsHavePerformancesServiceImpl.java
ackend/service/impl/ArtistServiceImpl.java
:end/service/impl/AddressServiceImpl.java
impl/SimpleMessageService.java
impl/MerchandiseServiceImpl.java
impl/SimpleNewsService.java
2020-04-16
2020-04-18
2020-04-20
2020-04-22
2020-04-24
2020-04-26
2020-04-28
2020-04-30
2020-05-02
2020-05-04
2020-05-06
2020-05-08
2020-05-10
2020-05-12
2020-05-14
2020-05-16
2020-05-18
2020-05-20
Go one level up
Author Mode
Co
☑student1@example.org 🟦
☑student2@example.org 🟧
☑student3@example.org 🟩

Current path: backend/src/main/java/at/ac/tuwien/sepm/grou

From 04/16/2020 To 06/22/202

Figure 5.11: *Author Mode* in a more detailed view.

5.4.3 Contribution Mode

Another visualization mode that was implemented based on the feedback during one of the pilot evaluations (see Subsection 6.1.1) was the *Contribution Mode*. In this mode, the contributions of students are displayed in an aggregated way. Each contributor is assigned the same color as in *Author Mode*.

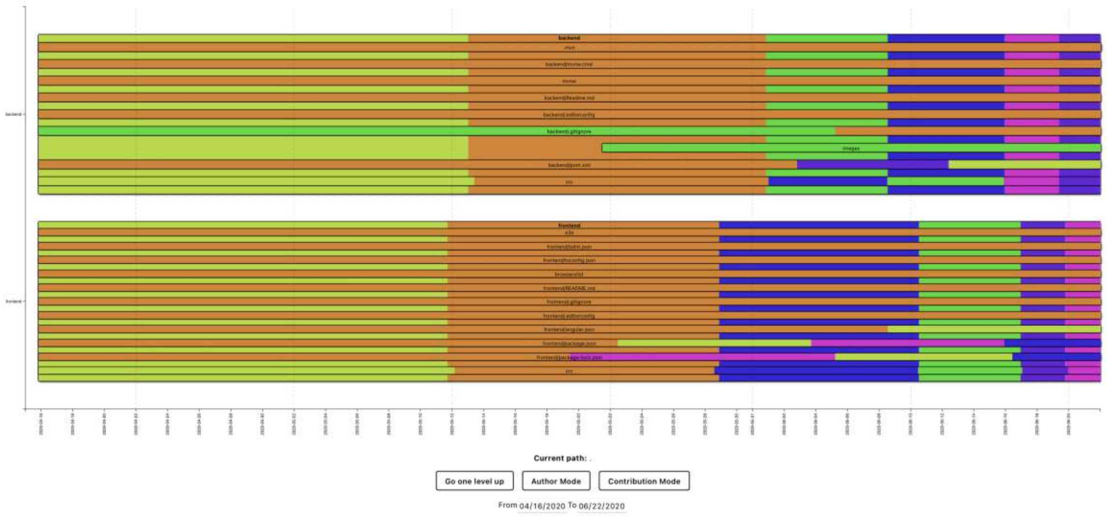


Figure 5.12: *Contribution Mode* in a top-level view.

The difference is, that each of the storylines is filled up with these colors based on the amount of contributions made by that person, as can be seen in Figure 5.12 and Figure 5.13. When hovering over a storyline, the amount of contributions by that person is displayed in both absolute and relative numbers.

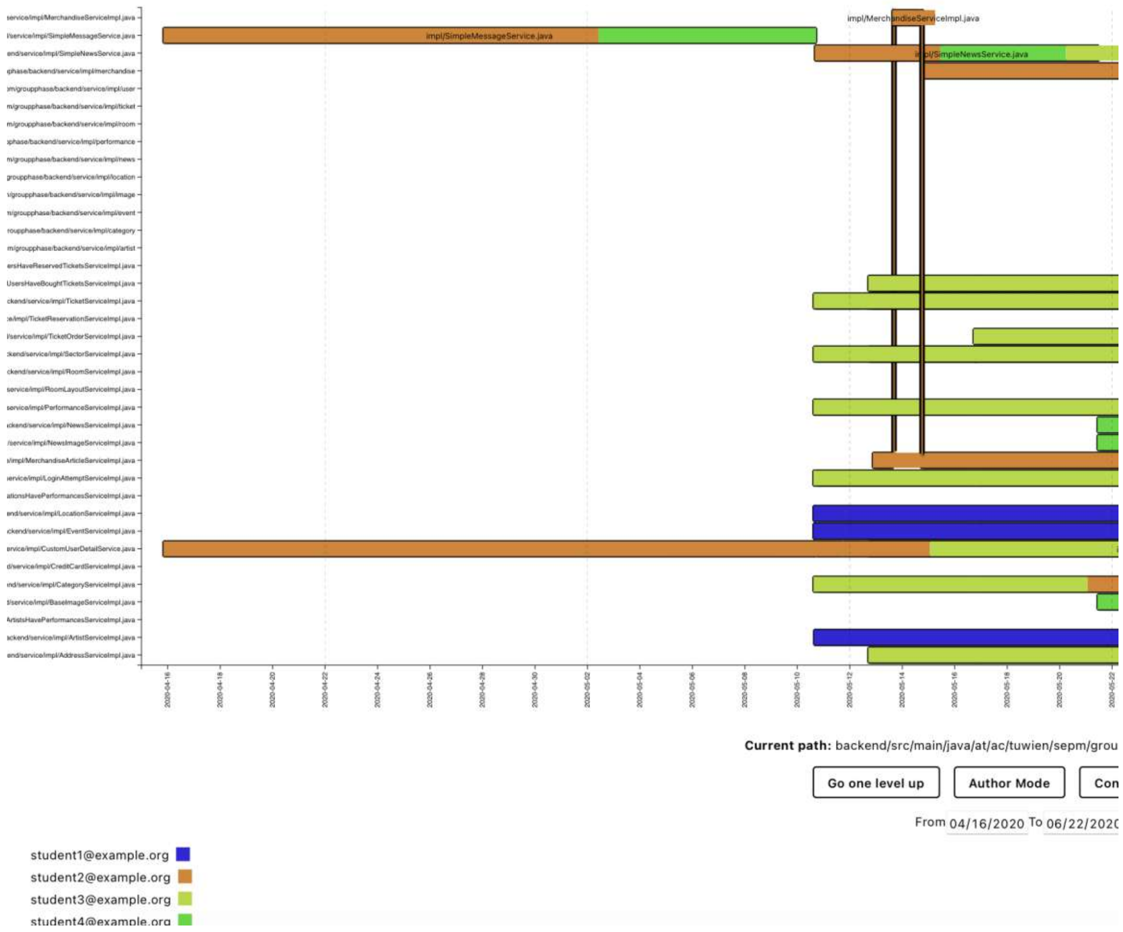

impl/MerchandiseServiceImpl.java
impl/SimpleMessageService.java
impl/SimpleNewsService.java
Current path: backend/src/main/java/at/ac/tuwien/sepm/grou
Go one level up
Author Mode
Con
From 04/16/2020 To 06/22/202
student1@example.org 🟦
student2@example.org 🟧
student3@example.org 🟩
student4@example.org 🟩

Figure 5.13: *Contribution Mode* in a more detailed view.

5.4.4 Graph Rendering

The frontend leverages the D3.js (Data-Driven Documents) [89] library in combination with React.js [85] and TypeScript [86] to render dynamic storylines. The data received from the backend is bound directly to Document Object Model (DOM) elements and rendered using Scalable Vector Graphics (SVG) elements.

The x-axis of the graph displays the time-frame of the visualized software evolution. This time-frame can be dynamically adjusted by using the date pickers underneath, causing the storylines to adapt automatically. The y-axis shows the paths of the top-level folders at the currently navigated level. Individual storylines are labeled with their respective names directly on the lines, allowing users to see the full path of each directory or file displayed.

The user can see whether a file or directory was added or deleted by observing where a storyline started or ended. Also, movement operations are visualized by vertically running parts of a storyline. Modifications to files at a certain point in time are indicated by areas with higher contrast on the storylines themselves.

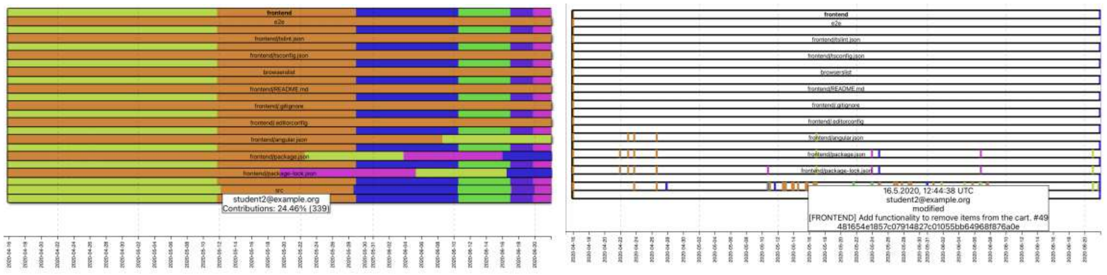
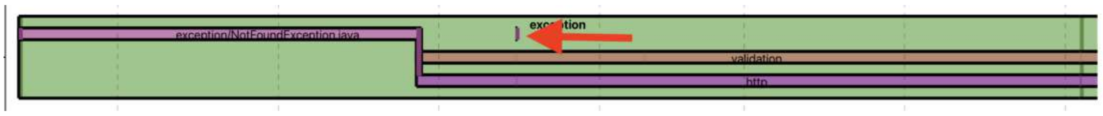


Figure 5.14: Hovering behavior in different modes.

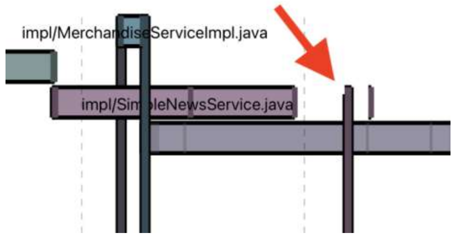
In the *Author Mode* or the *Contribution Mode*, hovering over elements such as modification indicators, reveals further information such as contributor information or details (see Figure 5.14).

5.4.5 Considerations

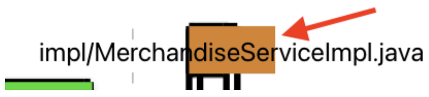
The frontend visualization was implemented with the intent to explore the idea of the storyline visualization as proposed in Subsection 4.1.1. The implementation should be seen as a prototype, which means that visual glitches or User Interface (UI)/User Experience (UX) limitations are to be expected. For example, some visual artifacts occurred with the dataset used for the the prototype evaluation (see Figure 5.15 for some examples).



(a) An already ended storyline reappears after movement of the file.



(b) A storyline is interrupted before indicating movement of the file.



(c) A storyline is too short and the coloring overflows in *Contribution Mode*.

Figure 5.15: Some examples of visual glitches in the frontend.

Regarding UI/UX limitations, switching between modes can be confusing because each activated mode has to be toggled off individually. This means that if both *Author Mode* and *Contribution Mode* get activated, both have to be turned off again by clicking both buttons again in order to return to the default mode. Some feedback regarding the UI/UX limitations was also received in the prototype evaluation (see Subsection 6.2.8).

5.5 Shared Library

Since both the frontend module and backend module were built using TypeScript [86], it became apparent during the implementation that types are duplicated on both sides. As the frontend uses types that are generated by the backend, the idea of a shared package utilized by both parts of the project came up.

In order to achieve this, a third module was created, called `shared-library`. It contains a `types.ts`, which exports enums, types, interfaces and type guards that are used across the project. Later on, a util function for conveniently adding entries to maps was also added.

Using the linking abilities offered by NPM, this package was then linked in both frontend and backend. This reduced code duplication and the effort required to change type definitions in the prototype project.

5.6 Artifacts

The complete source-code was provided in a Git repository. The code was licensed under the MIT license. The exact commit hash used in the evaluation was the following:

`8d29d20a6898eae915634a5b44f79f2774913c2c`.

5.6.1 Evaluation Environment

For the deployment used for the evaluation of the prototype, only the frontend itself was deployed as a static page on Netlify. This then loaded a pre-mined JSON file of the sample repository history at runtime. The reason for that was that the mining process would have taken too long to be done on the fly. For building and deploying, Netlify's Ubuntu 24.04 build image with Node.js v22.19.0 was used. For reproduction purposes, the prototype can also be run locally by cloning the project and running `npm ci` and `npm run start`. The pre-mined JSON (`test_scenario.json`) was included in the version controlled source code.

Additionally, the following configuration was used during the prototype evaluation:
Browser: Chrome 126
Hardware: Windows 10 Laptop with 8GB RAM

CHAPTER 6

Results

The storyline visualization technique that was selected in Section 4.2, was implemented and evaluated in multiple iterations.

6.1 Prototype Evaluation Design

A within-subject mixed-method evaluation [14][69] was conducted in the next step, in which six professionals in the fields of software engineering and education were first asked demographic questions and then tasked with completing 13 tasks in four scenarios twice to compare the developed prototype directly against IntelliJ IDEA. After each completion of the scenarios, they were asked to complete a set of post-task questions.

IntelliJ IDEA with the latest version (2025.2) and no additional plugins was selected as the state-of-the-art tool to compare the prototype against since it is widely used by the Java developer community as the literature and surveys among developers acknowledge [90] [52]. Monteiro et al. state that IntelliJ IDEA has been highly popular in recent years and developed an IntelliJ IDEA plugin partly for that reason [52]. According to the Snyk JVM Ecosystem Report, „Over 70% of JVM developers use IntelliJ IDEA" [92]. Baeldung also reports in their *Java IDEs in 2024* survey that IntelliJ IDEA „is the most popular Java IDE" [90]. Another reason for that choice was that the example repository used for the evaluation (see Subsection 6.1.2) was a project that used Java [84] and Angular [81].

6.1.1 Pilot Evaluations

Before starting with the prototype evaluations, two pilot evaluations were performed. For these preliminary tests, a person separate from the pool of participants was chosen. During these sessions, the candidate was asked to complete the scenarios (see Subsection 6.1.5) with both tools just like in the real prototype evaluation under the same conditions.

The purpose was to find a balanced task completion time limit which did not restrict participants too much in their successful completion of their task, but also did not provide an excessive amount of time as the overall time windows of the evaluations were limited.

In the pilot sessions, the average completion time per scenario was approximately 60-120 seconds for both tools, with some variation depending on the complexity of the tasks. Based on these results, the final time limit for each scenario was set to 120 seconds, which was supposed to make it possible to complete most tasks with a small buffer.

Since the main goal was to test timing and clarity, rather than collecting performance data, using a single participant was deemed to be sufficient. Additionally, due to some confusion during the pilot sessions, some of the task descriptions were clarified, but no major adjustments were necessary.

6.1.2 Dataset

Before designing the scenarios and tasks described in Subsection 6.1.5, a dataset had to be created that could be used for the evaluation. The requirements were that it provided both realistic data, while limited in size. For this purpose, the software repository of a real-world student project developed by a student group during a software engineering course at TU Wien was analyzed and replicated.

The repository data then went through the mining pipeline described in Section 5.3 and was used by the frontend module (see Section 5.4) during the prototype evaluation.

6.1.3 Demographics & Tool Familiarity

The demographic questions target the professional role and years of experience. Additionally, the participants were asked to gauge their familiarity with various tools. The following questions were asked:

Demographic Questions

1. **DQ1**: What is your role closest to?
2. **DQ2**: How many years of professional experience do you have?

Tool Familiarity

1. **TQ1**: How familiar are you with visualization tools for software repositories?
2. **TQ2**: How familiar are you with the Git CLI?
3. **TQ3**: How familiar are you with Git in general?
4. **TQ4**: How familiar are you with IntelliJ IDEA, especially its Git integration?

6.1.4 Tool Ordering

The design described in this chapter allowed for a within-subject comparison between the prototype and IntelliJ IDEA while minimizing potential biases due to the order in which the tools are used by participants to complete the scenarios.

Participant	Round 1	Round 2
1	Prototype	IntelliJ IDEA
2	Prototype	IntelliJ IDEA
3	IntelliJ IDEA	Prototype
4	IntelliJ IDEA	Prototype
5	Prototype	IntelliJ IDEA
6	IntelliJ IDEA	Prototype

Table 6.1: Tool ordering for each participant for completing the scenarios

As recommended by Sauro and Lewis for such a comparison, the starting tool was alternated between the prototype and IntelliJ IDEA (see Table 6.1). Three participants started with the prototype while the other three completed their tasks first using IntelliJ IDEA [69].

6.1.5 Scenarios & Tasks

For the evaluation of the prototype visualization, a set of scenarios was designed, each consisting of multiple tasks to be completed by the participants. These were chosen based on two considerations. First, they build directly onto the results found in the mockup evaluation (see Section 4.3), where participants highlighted how useful visualizing of operations, author contributions, refactorings and code frequency was perceived. Second, the tasks were aligned with the research questions defined in Section 7.1, such that they cover the proposed benefits: project transparency, refactoring frequency, code hotspots, and stale code detection. This approach adheres to practices in the field of educational software engineering, where the transparency of contributions are seen as important for tutors and student assistants [48] [26].

Each participant was given four scenarios with multiple tasks to complete once using the developed prototype and once using IntelliJ IDEA using the ordering described in Subsection 6.1.4. The same scenarios and tasks were used for each tool and participant during the evaluation.

Before starting each evaluation, a participant would be introduced to the scenarios with the following information:

> You are a tutor in a software engineering course at the University of Technology Vienna. The group you are tutoring has to submit their project in 2 weeks, it's the 22nd of June 2020. You are preparing for tomorrow's weekly jour fixe.

The following four scenarios were chosen for the evaluation. Each consists of multiple tasks that are related to each other and their scenario. The tasks are presented here as they were shown to the participants during the evaluation. Additionally, we specify what constituted the successful or partially successful completion of each task.

Scenario 1: Folder Analysis

Last week the group told you about the **admin-artists** feature they worked on. They started implementing the frontend components within the *frontend/src/app/components/admin-components/admin-artists* folder for this feature two weeks ago.

1. On which day was the folder created?
    - **Success**: 2020-06-07 is named
2. When was the latest contribution made within this folder?
    - **Success**: 2020-06-21, 19:31:50 is named
3. Who are the contributors of that folder?
    - **Success**: student1 and student3 are named

Scenario 2: Individual Student Contribution

One of your students, **student6**, seems to be behind in their tracked contribution time. Analyze their contributions over the last two months.

1. Focus on the *frontend/src/app/components* folder. To which direct sub-folders (so limited to the same level) did **student6** contribute?
    - **Success**: *ticket-components*, *user-components*, *news-components*, *event-components*, *admin-components* are named
    - **Partial Success**: One sub-folder is missed
2. In which of these sub-folders did **student6** do the most contributions in absolute numbers?
    - **Success**: *admin-components* is named
3. Now focus on the root directory. How many contributions did **student6** create for both the frontend and backend folders?
    - **Success**: 47 for frontend and 92 for backend
4. Which students seem to contribute significantly less to the whole project than other students?

- **Success**: student5 and student6 are named

5. Identify the top 3 contributing students for the **backend** folder.

   - **Success**: student2, student3 and student4 are named

Scenario 3: Refactor Analysis

One topic at last week's jour fixe was the refactoring made in the *backend/src/main/-java/at/ac/tuwien/sepm/groupphase/backend/service/impl* folder. In order to prepare revisiting this discussion again, you want to check the following.

1. What happened to the *NewsServiceImpl.java* and *MerchandiseArticleServiceImpl.java* files over the course of development with regard to creations, renames, movements or deletions?

   - **Success**: *NewsServiceImpl.java* was first created as *SimpleNewsService.java* and then renamed to *NewsServiceImpl.java*. *MerchandiseArticleServiceImpl.java* was first created as *MerchandiseArticleServiceImpl*, renamed to *Merchan-diseServiceImpl.java*, and then renamed again to *MerchandiseArticleServi-ceImpl.java*.
   - **Partial Success**: If one of the two services is correctly described

2. In which time-frame did these contributions occur?

   - **Success**: *NewsServiceImpl*: 2020-05-11 - 2020-06-21, *MerchandiseArticleSer-viceImpl*: 2020-05-13 - 2020-06-21
   - **Partial Success**: If they are off by ± 2 days

3. Who was involved in the contributions?

   - **Success**: student2, student3 and student4 are named

Scenario 4: Code Frequency Analysis

You would like to explore the frequency of contributions in the student project.

1. Identify rough time-frames when the folders *frontend/src* and *backend/src* were most frequently contributed to.

   - **Success**: *frontend/src/*: 2020-05-11 - 2020-05-17, 2020-05-22 - 2020-06-08, *backend/src/*: 2020-05-10 - 2020-05-18, 2020-05-24 - 2020-06-09
   - **Partial Success**: If they are off ± 2 days

2. Which 3 of the sub-folders 2 levels below the *frontend/src* folder (e.g. *frontend/sr-c/app/components*, *frontend/src/app/dtos*, ...) are the most heavily modified?

   - **Success**: *frontend/src/app/components*, *frontend/src/app/services*, *frontend/s-rc/app/dtos* are named

6.1.6 Evaluated Metrics

During the completion of the above described tasks by the participants, the following metrics were evaluated:

Task Success Rate

The percentage of tasks successfully completed was calculated to determine the success rate of the tasks. Each task was rated as either a success, partial success, or a failure. Clear criteria were established to define what constituted a successful task completion, as can be seen in Subsection 6.1.5.

Completion Time

The time taken to complete each task was recorded to measure efficiency. The completion time of the task was measured from the moment the task started until it was successfully completed.

Perceived Difficulty

After each task, the participants were asked to rate how difficult they perceived it on a scale of 1-5.

6.1.7 SUS Scoring

After completing the tasks with each tool, the participants were asked to complete a short survey consisting of ten Likert-scale responses [38]. The purpose of the questionnaire was to determine the SUS Scoring of both the prototype and IntelliJ IDEA:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

6.1.8 Qualitative Feedback

After completing the evaluation, participants were asked a set of post-task questions, comparing the prototype with IntelliJ IDEA and naming any noticed advantages or disadvantages. These questions were answered verbally, transcribed and finally analyzed to complement the quantitative findings. The following questions were asked:

- **PQ1**: In comparison with IntelliJ IDEA, what advantages or disadvantages do you see when using the prototype?
- **PQ2**: Separately from that comparison, are there any disadvantages you noticed about the prototype itself?
- **PQ3**: In comparison with the prototype, what advantages or disadvantages do you see when using IntelliJ IDEA?
- **PQ4**: Separately from that comparison, are there any disadvantages you noticed about IntelliJ IDEA itself?

We label these questions with **PQ1-PQ4** in order to refer to them in subsequent chapters.

6.1.9 Task-Benefit Mapping

The research question asked in Section 1.2 targets benefits, namely *Project Transparency*, *Refactoring Frequency*, *Code Hotspots* and *Stale Code Detection*. We mapped the tasks presented in Subsection 6.1.5 to these benefits. For each of the metrics *Task Success Rate*, *Completion Time* and *Perceived Difficulty* (see Subsection 6.1.6), the results in Chapter 6 were grouped by each participant for all tasks and also for each benefit.

The mapping was made as shown in Table 6.2.

As can be seen in Table 6.2, most of the tasks in the evaluation were mapped to the *Project Transparency* benefit, with not as many tasks covering *Refactoring Frequency* and *Code Hotspots & Stale Code Detection*. This was done on purpose, as the focus of this thesis was put on the educational context, where understanding individual and group contributions plays an important role for tutors and student assistants [48] [70] [25] [50].

The focus on *Project Transparency* allowed for a more detailed analysis of that benefit, but it may have also influenced the aggregated results across all tasks. For this reason, the interpretation of the evaluation results in Chapter 7 was done separately for each benefit, with the acknowledgment that findings for *Refactoring Frequency* and *Code Hotspots & Stale Code Detection* were based on fewer tasks and were therefore less comprehensive.

Task	Benefit
Folder Analysis 1	Project Transparency
Folder Analysis 2	Project Transparency
Folder Analysis 3	Project Transparency
Individual Student Evaluation 1	Project Transparency
Individual Student Evaluation 2	Project Transparency
Individual Student Evaluation 3	Project Transparency
Individual Student Evaluation 4	Project Transparency
Individual Student Evaluation 5	Project Transparency
Refactor Analysis 1	Refactoring Frequency
Refactor Analysis 2	Refactoring Frequency
Refactor Analysis 3	Refactoring Frequency
Code Frequency Analysis 1	Code Hotspots & Stale Code Detection
Code Frequency Analysis 2	Code Hotspots & Stale Code Detection

Table 6.2: Task-Benefit Assignment in the Evaluation

6.2 Prototype Evaluation Results

In this section, the results of the evaluation described in Section 6.1 are presented. After shortly describing the method used for calculating statistical significance of the results, we analyze the evaluation data by focusing on the chosen metrics (see Subsection 6.1.6).

6.2.1 Statistical Significance

In the following subsections, we determined the statistical significance of the presented results. This was done using paired t-tests, which can be used for within-subject comparisons such as this evaluation:

$$t = \frac{\hat{D}}{\frac{s_D}{\sqrt{n}}}$$

. For calculating the required p-value, the Microsoft Excel formula TDIST($|t|$, $(n-1)$, 2) was used [68].

In order to also determine the effect size of the difference between the results, we calculated Cohen's d for correlated measurements (d_z) directly from the t value as described by Lakens:

$$d_z = \frac{t}{\sqrt{n}}$$

[46]. The 95% Student- t confidence interval for the mean differences was calculated using the following formula as shown by Sauro and Lewis:

$$\hat{D} \pm t_a \frac{s_D}{\sqrt{n}}$$

$\hat{D}$ being the mean of the differences, n the number of participants, s_D the standard deviation of the difference scores and t_a the critical value of the t-distribution for $n-1$ degrees of freedom for a 95% confidence interval [68]. We calculate t_a using the following Microsoft Excel formula: T.INV.2T(0.05, 5).

6.2.2 Demographics

Six professionals participated in the prototype evaluation, which had a variety of roles. There was one university assistant, three software developers, a project manager and a tutor. That means half of the participants were software developers while the rest was either in managerial or teaching positions. The years of professional experience varied between 3 and 11 years. Two of them had 9-11 years of experience, three had 3-5 years and one participant 6-8 years of experience.

To also take the effects of tool proficiency into account, the participants were also asked about their familiarity with visualization tools for software repositories, the Git CLI, Git in general, and IntelliJ IDEA with its Git integration. The results of that can be found in Subsection 6.2.3.

	P1	P2	P3	P4	P5	P6
Role	Univ. Asst.	Dev.	PM	Dev.	Tutor	Dev.
Years Experience	9-11	9-11	3-5	3-5	3-5	6-8

Table 6.3: Demographics of participants

The sample presented in Table 6.3 included participants with a balanced variety of professional backgrounds to ensure that the evaluation results consider the perspectives of different professional contexts in both software engineering and software engineering education.

6.2.3 Evaluation of Tool Familiarity

In order to get an understanding on how skilled the participants were with the used tools, they were asked to answer questions on their familiarity with IntelliJ IDEA, Git, the Git CLI and visualization tools for software repositories. This data enabled more accurate interpretation of the evaluated metrics and the user experience results. They were asked these questions before starting the evaluation. We differentiate between two groups: participants that started with the prototype and those who began with IntelliJ IDEA. As can be observed in Figure 6.1 and Table 6.4, Participants of the first group had a higher mean familiarity with Git in general (4.0 vs. 3.33) and with the Git CLI (4.0 vs. 2.67). Those that started with IntelliJ IDEA showed a slightly higher mean familiarity with IntelliJ IDEA itself (2.67 vs. 2.0). The familiarity with visualization tools was the same for both groups with a mean of 3.0.

When looking at the standard deviations in Figure 6.2, the group that started with IntelliJ IDEA showed a greater variability with their ratings for familiarity with IntelliJ

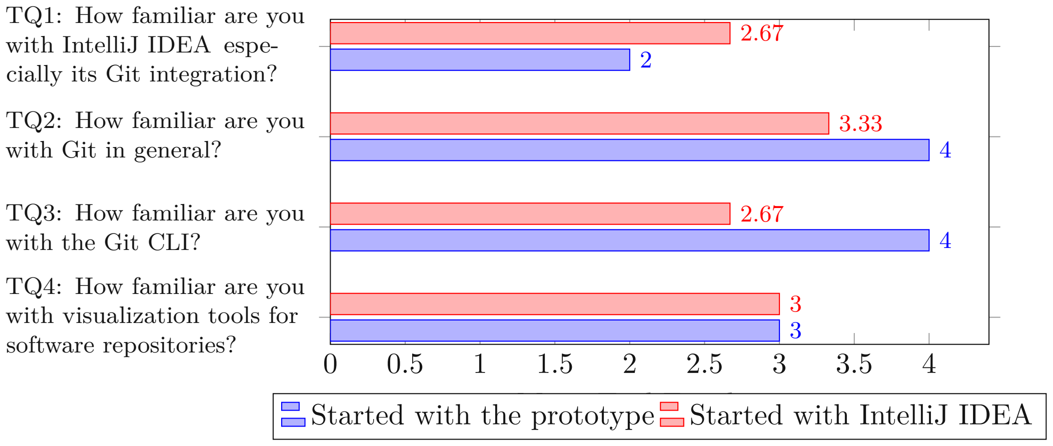


Figure 6.1: The mean of familiarity of participants with the tools that started with the prototype and also of the ones that started with IntelliJ IDEA (rounded to two decimal places).

Question	Prototype	IntelliJ IDEA	Δ = Prototype - IntelliJ IDEA
TQ1	3.0	3.0	0
TQ2	4.0	2.67	+1.33
TQ3	4.0	3.33	+0.67
TQ4	2.0	2.67	-0.67
Mean	**3.25**	**2.92**	**+0.33**

Table 6.4: The mean values of differences in tool familiarity ratings between the tools used to start the evaluation (refer to Subsection 6.1.3 for the question references).

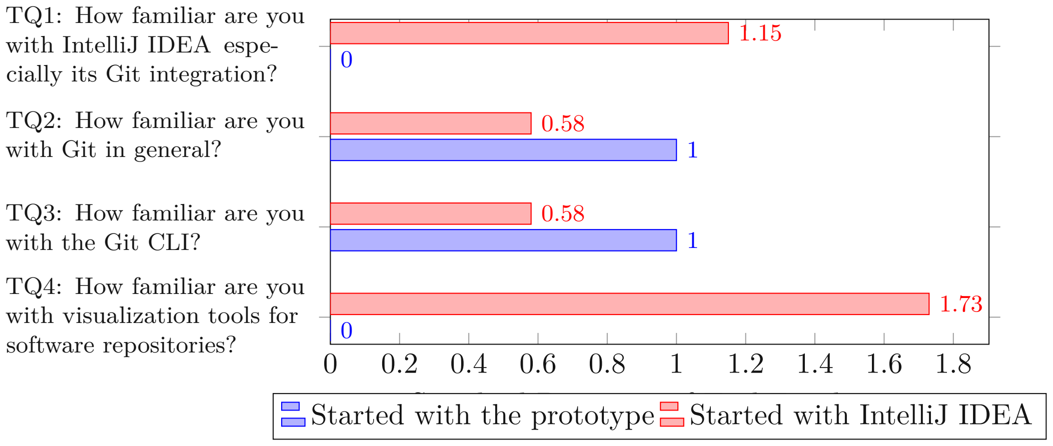


Figure 6.2: The standard deviation of familiarity with the tools of participants that started with the prototype and also of the ones that started with IntelliJ IDEA.

IDEA and its Git integration (1.15), while the other group showed no deviation at all. The same goes for familiarity with visualization tools for software repositories. Here, the IntelliJ IDEA group showed a variability of 1.73 vs. 0 for the prototype group.

The differences between the two groups were not extreme, but the variations in tool familiarity, especially participants that started with the prototype being more proficient with Git, are factors to be considered when interpreting the differences in success rates, perceived difficulty and completion times in Chapter 7.

6.2.4 Evaluation of Success Rate

When comparing the success rates between the developed prototype and IntelliJ IDEA (see Figure 6.3), the former had a slightly higher success rate, with approximately two thirds (66.7%) of tasks being successfully completed. Successful completions using IntelliJ IDEA were at 56.4%. When looking at the corresponding failure rate, it can be seen that it was lower for the prototype (32.1%) when compared to IntelliJ IDEA (42.3%). Partial successes were rare for both tools with an identical rate of 1.3%. In Figure 6.4

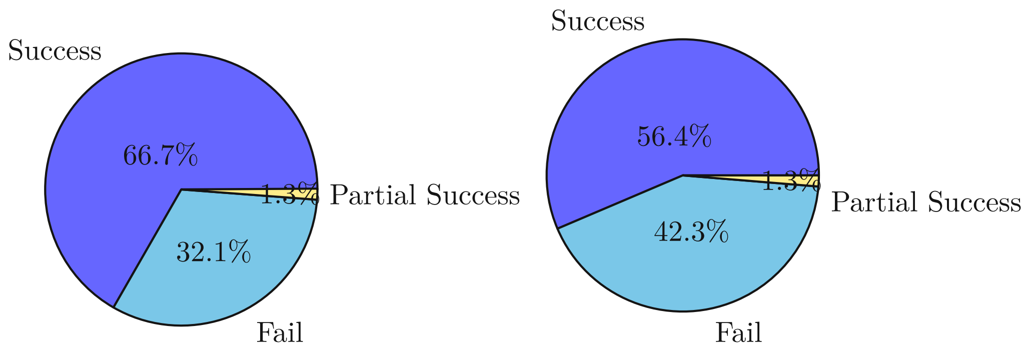


Figure 6.3: Success Rates for the Prototype and IntelliJ IDEA.

and Figure 6.5, it can be observed that the prototype achieved a perfect success rate of 100% for several tasks („Folder Analysis 2“, „Folder Analysis 3“, „Individual Student 1“, „Individual Student 4“), while having a very low success rate for the tasks „Individual Student 2“ and „Refactor Analysis 2“. Another notable result is the success rate of „Refactor Analysis 1“, which was rarely completed with a partial success, while never being fully successfully done. The results of IntelliJ IDEA are more varied and indicate a lower success rate across the tasks, namely „Individual Student 1“, „Individual Student 3“, „Code Frequency 1“ and „Code Frequency 2“. Similar to the prototype, „Refactor Analysis 1“ was never completed successfully, not even partially. As with the prototype, the task „Folder Analysis 2“ showed a perfect success rate of 100%.

When looking at the success rates of the individual participants for both the prototype and IntelliJ IDEA in Table 6.5, we see that there is a slightly higher mean success rate on the former for most of the participants. The only exception is participant 5, who had a higher success rate when completing the tasks using IntelliJ IDEA.

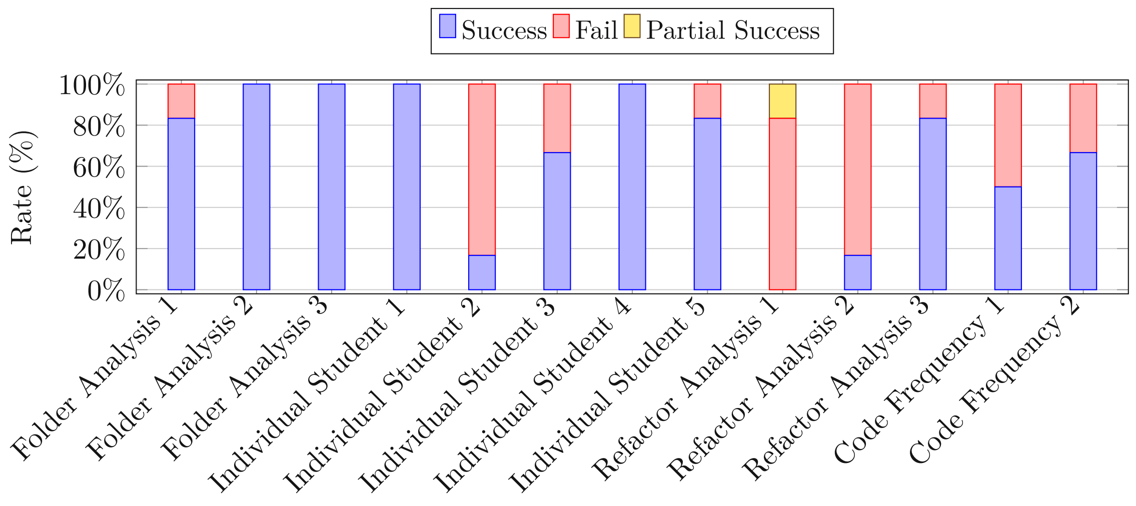


Figure 6.4: Success Rates by Task for the Prototype.

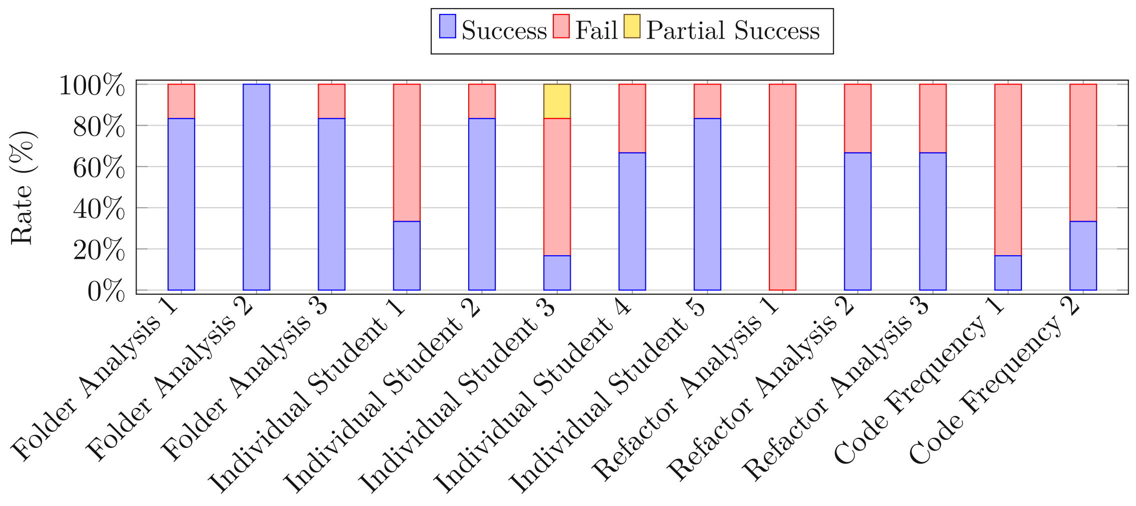


Figure 6.5: Success Rates by Task for IntelliJ IDEA.

In order to determine whether the overall difference between the tools in this regard was statistically significant, we performed t-tests as described in Subsection 6.2.1 for each benefit and across all tasks.

Success Rate per Participant

For the differences between participant success rates across all tasks, as shown in Table 6.5, the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{0.102}{\frac{0.221}{\sqrt{6}}} = 1.135, \quad (p = 0.307)$$

The p-value amounted to $0.307 > 0.05$. The difference between success rates of the tools was not statistically significant. The 95% confidence interval ranged from -0.130 to 0.335 [69].

Cohen's d for correlated measurements (d_z) amounted to:

$$d_z = \frac{1.135}{\sqrt{6}} = 0.464$$

This small to medium effect size [46] when comparing the tools across every task indicated

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	76.92%	61.53%	+15.38%
Participant 2	61.53%	53.84%	+7.69%
Participant 3	84.61%	38.46%	+46.15%
Participant 4	69.23%	61.53%	+7.69%
Participant 5	46.15%	69.23%	-23.08%
Participant 6	61.53%	53.84%	+7.69%
Mean	**66.67%**	**56.41%**	**+10.26%**

Table 6.5: Per-participant success rates for all tasks including their differences (rounded to two decimal places).

that participants' success rates were on average about 0.464 standard deviations higher when using the prototype.

Project Transparency: Success Rate per Participant

For the differences between participant success rates for the *Project Transparency* benefit, as shown in Table 6.6, the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{0.125}{\frac{0.224}{\sqrt{6}}} = 1.369, \quad (p = 0.229)$$

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	87.50%	62.50%	+25.00%
Participant 2	75.00%	75.00%	+0.00%
Participant 3	100.00%	62.50%	+37.50%
Participant 4	87.50%	62.50%	+25.00%
Participant 5	62.50%	87.50%	–25.00%
Participant 6	75.00%	62.50%	+12.50%
Mean	**81.25%**	**68.75%**	**+12.50%**

Table 6.6: Per-participant success rates for the Project Transparency benefit.

The p-value amounted to $0.229 > 0.05$. The difference between success rates of the tools was not statistically significant. The 95% confidence interval ranged from -0.110 to 0.360 [69].

Cohen's d for correlated measurements (d_z) amounted to:

$$d_z = \frac{1.369}{\sqrt{6}} = 0.559$$

This medium effect size [46] on *Project Transparency* related tasks indicated that the prototype had a better success rate by over half a standard deviation when compared to IntelliJ IDEA.

Refactoring Frequency: Success Rate per Participant

When looking at the differences in success rates for the *Refactoring Frequency* benefit (see Table 6.7), the corresponding p-value calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-0.222}{\frac{0.172}{\sqrt{6}}} = -3.162, \quad (p = 0.025)$$

The p-value amounted to $0.025 < 0.05$.

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	33.33%	66.67%	-33.33%
Participant 2	33.33%	33.33%	0.00%
Participant 3	33.33%	33.33%	0.00%
Participant 4	33.33%	66.67%	-33.33%
Participant 5	33.33%	66.67%	-33.33%
Participant 6	33.33%	66.67%	-33.33%
Mean	**33.33%**	**55.55%**	**-22.22%**

Table 6.7: Per-participant success rates for the Refactoring Frequency benefit.

The difference between success rates of the tools was statistically significant. The 95% confidence interval ranged from -0.403 to -0.042 [69].

Cohen's d for correlated measurements (d_z) amounted to:

$$d_z = \frac{-3.162}{\sqrt{6}} = -1.290$$

This meant a large effect size [46] by which IntelliJ IDEA did better than the prototype on refactoring-frequency tasks.

Code Hotspots & Stale Code Detection: Success Rate per Participant

For differences in success rates for the *Code Hotspots & Stale Code Detection* (see Table 6.8), the corresponding p-value calculated as described in Subsection 6.2.1:

$$t(5) = \frac{0.1667}{\frac{0.258}{\sqrt{6}}} = 1.581, \quad (p = 0.175)$$

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	100.00%	50.00%	+50.00%
Participant 2	50.00%	50.00%	0.00%
Participant 3	100%	50.00%	+50.00%
Participant 4	50.00%	50.00%	0.00%
Participant 5	50.00%	50.00%	0.00%
Participant 6	50.00%	50.00%	0.00%
Mean	**66.67%**	**50.00%**	**+16.67%**

Table 6.8: Per-participant success rates for the *Code Hotspots & Stale Code Detection* benefits.

The p-value amounted to $0.175 > 0.05$. The difference between success rates of the tools was not statistically significant. The 95% confidence interval ranged from -0.104 to 0.437 [69].

Cohen's d for correlated measurements (d_z) amounted to:

$$d_z = \frac{1.581}{\sqrt{6}} = 0.645$$

This indicated a medium to large effect size [46]. For this benefit, the prototype performed better than IntelliJ IDEA by two-thirds of a standard deviation.

Summary: Success Rate per Participant

Overall, only *Refactoring Frequency* showed a statistically significant advantage for IntelliJ IDEA.

Benefit	$\bar{D}$	S_D	$t(5)$	p	d_z	Significant
All tasks	0.102	0.221	1.14	0.31	0.46	no
Project Transparency	0.125	0.224	1.37	0.23	0.56	no
Refactoring Frequency	–0.222	0.172	–3.16	0.025	–1.29	yes
Code Hotspots & Stale Code	0.167	0.258	1.58	0.18	0.65	no

Table 6.9: Summarized success-rate comparisons (n=6, df=5).

The prototype performed better in the *Project Transparency* and *Code Hotspots & Stale Code Detection* benefits, which had statistically non-significant, medium to large effect sizes (see Table 6.9).

6.2.5 Evaluation of Perceived Difficulty

During the evaluation, participants were asked after each task to rate how difficult they perceived it on a scale from 1-5.

Completion Times per Participant

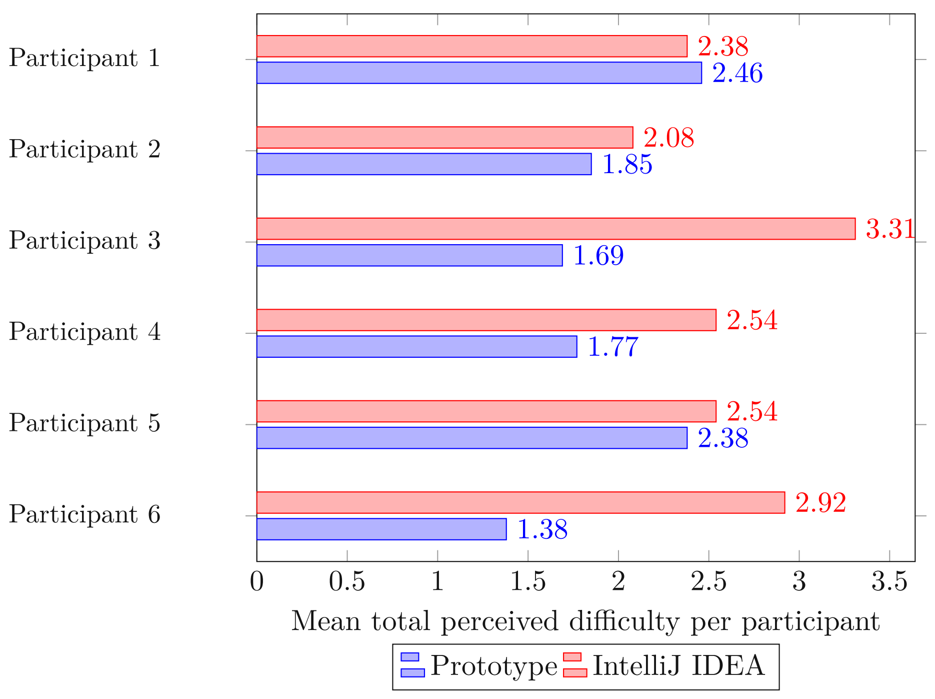


Figure 6.6: The mean of the perceived difficulties per participant for the tasks that were performed using both the prototype and IntelliJ IDEA.

Overall, the mean perceived difficulty of participants ($n = 6$), as presented in Figure 6.6, over all tasks is higher for IntelliJ IDEA than the prototype. One exception being Participant 1 whose mean of ratings is slightly higher than the one of IntelliJ IDEA.

When looking at the distribution of perceived difficulty ratings for both tools (see Figure 6.7), it can be seen that almost half of the tasks completed with the prototype were rated with the easiest difficulty, 45.5%. 31.2% of tasks were perceived with a difficulty of 2, while the difficulty ratings 3,4 and 5 were assigned to 11.7%, 7.8%, 3.9% of tasks respectively. The perceived difficulty ratings of IntelliJ IDEA were more varied. 29.9% of tasks were rated with the second-easiest difficulty, 26.0% with a difficulty rating of 3. 20.8% of tasks were perceived to be a difficulty rating of 1. The ratings 4 and 5 were given to 10.4% and 13.0% of tasks, respectively.

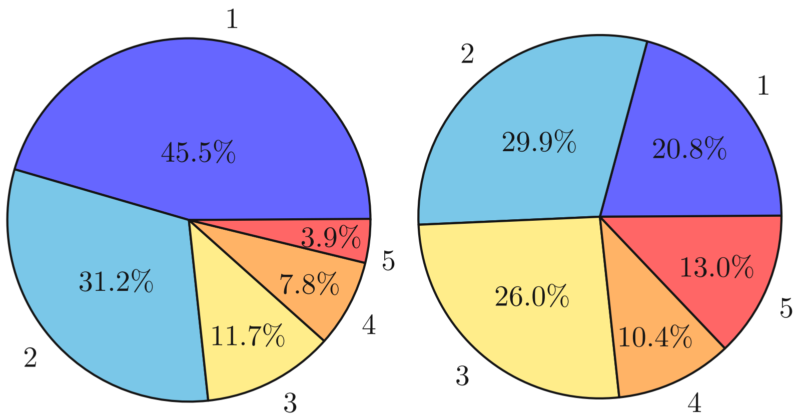


Figure 6.7: Distribution of perceived difficulty ratings for the prototype (left) and IntelliJ IDEA (right).

Perceived Difficulty per Participant

For the perceived difficulty ratings per participant across all tasks (see Table 6.10), the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-0.705}{\frac{0.730}{\sqrt{6}}} = -2.365, \quad (p = 0.064)$$

The p-value amounted to $0.064 > 0.05$. The difference between perceived difficulty ratings of the tools was not statistically significant. The 95% confidence interval ranged from -1.472 to 0.061 [69].

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	2.46	2.38	+0.08
Participant 2	1.85	2.08	–0.23
Participant 3	1.69	3.31	–1.62
Participant 4	1.77	2.54	–0.77
Participant 5	2.38	2.54	–0.15
Participant 6	1.38	2.92	–1.54
Mean	**1.92**	**2.63**	**-0.71**

Table 6.10: The mean values of differences in perceived difficulty ratings between the prototype and IntelliJ IDEA per participant (rounded to two decimal places).

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{-2.365}{\sqrt{6}} = -0.965$$

This value indicated a large effect size [46]. Across every task, participants' perceived difficulty ratings were on average about 0.965 standard deviations lower with the prototype.

Project Transparency: Perceived Difficulty per Participant

When looking at the perceived difficulty ratings with regard to the *Project Transparency* benefit (see Table 6.11), the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-0.979}{\frac{0.755}{\sqrt{6}}} = -3.173, \quad (p = 0.025)$$

The p-value amounted to $0.025 < 0.05$. The difference between perceived difficulty ratings of the tools was statistically significant. The 95% confidence interval ranged from -1.772 to -0.186 [69].

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	1.63	2.38	–0.75
Participant 2	1.75	1.50	+0.25
Participant 3	1.25	3.00	–1.75
Participant 4	1.13	2.50	–1.38
Participant 5	1.88	2.50	–0.62
Participant 6	1.13	2.75	–1.62
Mean	**1.46**	**2.44**	**-0.98**

Table 6.11: Perceived difficulty ratings for the Project Transparency benefit, per participant (rounded to two decimal places).

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{-3.173}{\sqrt{6}} = -1.295$$

This indicated a large effect size [46]. Across every task, participants' perceived difficulty ratings were on average about 1.295 standard deviations lower with the prototype.

Refactoring Frequency: Perceived Difficulty per Participant

Regarding the perceived difficulty ratings for *Refactoring Frequency*-related tasks (as can be seen in Table 6.12), the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-0.222}{\frac{1.409}{\sqrt{6}}} = -0.386, \quad (p = 0.715)$$

The p-value amounted to $0.715 > 0.05$. The difference between perceived difficulty ratings of the tools was not statistically significant. The 95% confidence interval ranged from -1.700 to 1.256 [69].

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{-0.386}{\sqrt{6}} = -0.158$$

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	3.67	2.00	+1.67
Participant 2	2.00	4.00	-2.00
Participant 3	2.00	3.67	-1.67
Participant 4	2.67	2.33	+0.33
Participant 5	3.33	2.67	+0.67
Participant 6	2.00	2.33	–0.33
Mean	**2.61**	**2.83**	**-0.22**

Table 6.12: Perceived difficulty ratings for the Refactoring Frequency benefit, per participant (rounded to two decimal places).

This indicated a small effect size [46], which suggests no reliable difference between perceived difficulty ratings.

Code Hotspots & Stale Code Detection: Perceived Difficulty per Participant

For *Code Hotspots & Stale Code Detection*-related tasks (see Table 6.13), the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-0.333}{\frac{1.472}{\sqrt{6}}} = -0.555, \quad (p = 0.603)$$

The p-value amounted to 0.603 > 0.05. The difference between perceived difficulty ratings of the tools was not statistically significant. The 95% confidence interval ranged from −1.878 to 1.211 [69].

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	4.00	3.00	+1.00
Participant 2	2.00	1.50	+0.50
Participant 3	3.00	4.00	–1.00
Participant 4	3.00	3.00	0.00
Participant 5	3.00	2.50	+0.50
Participant 6	1.50	4.50	-3.00
Mean	**2.75**	**3.08**	**-0.33**

Table 6.13: Perceived difficulty ratings for the Code Hotspots & Stale Code Detection benefits, per participant (rounded to two decimal places).

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{-0.555}{\sqrt{6}} = -0.226$$

This value indicated a small effect size [46], which suggests no reliable difference between perceived difficulty ratings.

Summary: Perceived Difficulty per Participant

Overall, only *Project Transparency* showed a statistically significant advantage for the prototype.

Benefit	$\bar{D}$	S_D	$t(5)$	p	d_z	Significant
All tasks	-0.705	0.730	-2.365	0.064	-0.965	no
Project Transparency	-0.979	0.756	-3.173	0.0247	-1.295	yes
Refactoring Frequency	-0.222	1.409	-0.386	0.715	-0.158	no
Code Hotspots & Stale Code	-0.333	1.472	-0.554	0.603	-0.226	no

Table 6.14: Summarized perceived difficulty comparisons (n=6, df=5).

It also performed better in the *Refactoring Frequency* and *Code Hotspots & Stale Code Detection* benefits, which had statistically non-significant, small effect sizes (see Table 6.14).

6.2.6 Evaluation of Completion Times

The completion time of each task for each participant was tracked by the moderator using a digital stopwatch. If during a task, the completion time of 120 seconds was reached, the task would be rated as failed and marked as a timeout. When evaluating the completion times, it was decided to omit the results that timed out to not skew the mean of the measurements.

Completion Times per Participant

Across all tasks, differences in completion times are shown in Table 6.15. The corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-11.603}{\frac{12.887}{\sqrt{6}}} = -2.205, \quad (p = 0.079)$$

The p-value amounted to $0.079 > 0.05$. The difference between completion times of the tools was not statistically significant. The 95% confidence interval ranged from -25.127 to 1.922 [69].

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{-2.205}{\sqrt{6}} = -0.900$$

This indicated a large effect size [46]. Across every task, participants' completion times were on average about 0.900 standard deviations lower with the prototype.

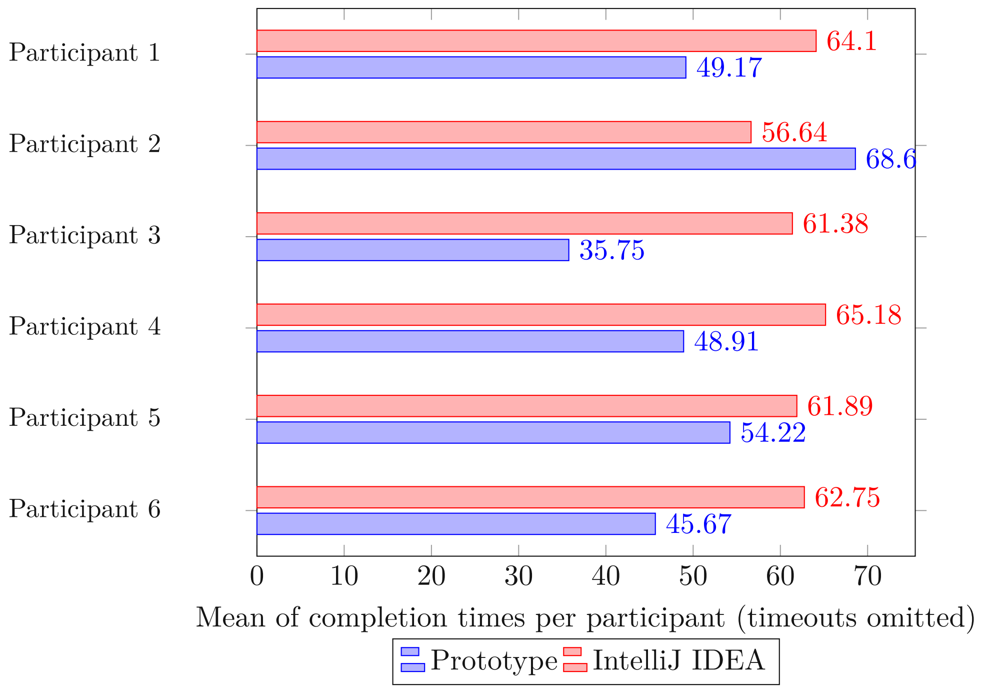


Figure 6.8: The mean of the completion times of the tasks for both the prototype and IntelliJ IDEA per participant.

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	49.17	64.1	-14.93
Participant 2	68.60	56.64	+11.96
Participant 3	35.75	61.37	-25.62
Participant 4	48.90	65.18	-16.27
Participant 5	54.22	61.89	-7.67
Participant 6	45.67	62.75	-17.08
Mean	**50.39**	**61.99**	**-11.60**

Table 6.15: The mean completion times including the differences between the prototype and IntelliJ IDEA per participant (in seconds, rounded to two decimal places).

Project Transparency: Completion Times per Participant

The differences in completion times for *Project-Transparency*-related tasks are shown in Table 6.16. The corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{-25.031}{\frac{18.005}{\sqrt{6}}} = -3.405, \quad (p = 0.019)$$

The p-value amounted to $0.019 < 0.05$. The difference between completion times of the tools was statistically significant. The 95% confidence interval ranged from -43.926 to -6.136 [69].

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	25.33	66.25	–40.92
Participant 2	62.67	52.43	+10.24
Participant 3	29.67	65.33	–35.67
Participant 4	28.67	67.14	–38.48
Participant 5	19.00	62.14	–43.14
Participant 6	31.33	62.80	–31.47
Mean	**37.65**	**62.68**	**-25.03**

Table 6.16: Mean completion times for the *Project Transparency* benefit, per participant (in seconds, rounded to two decimal places).

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{-3.405}{\sqrt{6}} = -1.390$$

This indicated a large effect size [46], which suggests a significant difference between the prototype and IntelliJ IDEA, the former having significantly lower completion times for the tasks of the *Project Transparency* benefit.

Refactoring Frequency: Completion Times per Participant

The differences in completion times for the *Refactoring Frequency* benfit can be seen in Table 6.17. The corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(5) = \frac{21.583}{\frac{24.953}{\sqrt{6}}} = 2.119, \quad (p = 0.088)$$

The p-value amounted to $0.088 > 0.05$. The difference between completion times of the tools was not statistically significant. The 95% confidence interval ranged from -4.603 to 47.770 [69].

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	91.00	55.50	+35.50
Participant 2	77.00	52.50	+24.50
Participant 3	32.50	49.50	-17.00
Participant 4	84.00	35.50	+48.50
Participant 5	98.50	61.00	+37.50
Participant 6	50.00	49.50	+0.50
Mean	**72.17**	**50.58**	**+21.59**

Table 6.17: Mean completion times for the *Refactoring Frequency* benefit, per participant (in seconds, rounded to two decimal places).

Cohen's d for correlated measurements (d_z) amounted to

$$d_z = \frac{2.119}{\sqrt{6}} = 0.865$$

This indicated a large effect size [46]. Across every task for the *Refactoring Frequency* benefit, participants' completion times were on average about 0.865 standard deviations lower with IntelliJ IDEA.

Code Hotspots & Stale Code Detection: Completion Times per Participant

For the differences in completion times regarding tasks targeting the *Code Hotspots & Stale Code Detection* benefit (see Table 6.18), the corresponding p-value was calculated as described in Subsection 6.2.1:

$$t(2) = \frac{10.500}{\frac{6.144}{\sqrt{3}}} = 2.960, \quad (p = 0.098)$$

The p-value amounted to $0.098 > 0.05$. The difference between completion times of the tools was not statistically significant. The 95% confidence interval ranged from -4.762 to 25.762 [69].

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
Participant 1	82.00	-	-
Participant 2	88.50	75.50	+13.00
Participant 3	78.50	-	-
Participant 4	103.00	88.00	+15.00
Participant 5	-	-	-
Participant 6	92.50	89.00	+3.50
Mean	**94.67**	**84.17**	**+10.50**

Table 6.18: Mean completion times for the *Code Hotspots & Stale Code Detection* benefits, per participant (in seconds, rounded to two decimal places).

In this case, we only count completion times for the participants 2,4 and 6, since only they were able to complete the respective tasks with both tools in time. Thus, $n = 3$ for this comparison.

Cohen's d for correlated measurements (d_z) amounted to:

$$d_z = \frac{2.960}{\sqrt{3}} = 1.709$$

This indicated a large effect size [46]. Across every task for the *Code Hotspots & Stale Code Detection* benefits, participants' completion times were on average about 1.709 standard deviations lower with IntelliJ IDEA.

Summarized Results

Overall, only *Project Transparency* showed a statistically significant advantage for the prototype.

Benefit	$\bar{D}$	S_D	$t(df)$	p	d_z	Significant
All tasks	-11.602	12.887	-2.21	0.0786	-0.90	no
Project Transparency	-25.031	18.004	-3.405	0.019	-1.390	yes
Refactoring Frequency	21.583	24.953	2.119	0.0876	0.865	no
Code Hotspots & Stale Code	10.50	6.144	2.960	0.0977	1.709	no

Table 6.19: Summarized completion time comparisons. df varies by row (All Tasks/Project Transparency/Refactoring Frequency: $df = 5$; Code Hotspots & Stale Code Detection: $df = 2$)

IntelliJ IDEA performed better in the *Refactoring Frequency* and *Code Hotspots & Stale Code Detection* benefits, which had statistically non-significant, large effect sizes (see Table 6.19).

6.2.7 Evaluation of System Usability Scale (SUS)

After working through the tasks, the participants ($n = 6$) were asked to complete a standardized questionnaire based on the System Usability Scale (SUS) for both the prototype and IntelliJ IDEA. The SUS is a Likert scale based on ten items that makes it possible to create a scoring between 0 and 100 on the overall subjective usability of a tool. A score is calculated by taking the scale positions from the questions 1,3,5,7, and 9 minus 1. The remaining scores from the other questions 2,4,6,8, and 10 are also incorporated by subtracting each of them from 5. This raw score is then multiplied by 2.5 to calculate the overall value for a participant: $(\sum_{n=1,3,5,7,9}(Q_n - 1) + \sum_{n=2,4,6,8,10}(5 - Q_n)) * 2.5$ [38] [11] [7]. This is done for the responses of each participant (see Figure 6.9). The calculated SUS scores of each participant for both tools including their differences can be seen in Table 6.20.

Participant	Prototype	IntelliJ IDEA	Δ vs. Prototype
1	75.0	42.5	+32.5
2	82.5	55.0	+27.5
3	92.5	52.5	+40.0
4	72.5	62.5	+10.0
5	75.0	67.5	+7.5
6	85.0	37.5	+47.5
Mean	**80.42**	**52.92**	**+27.5**

Table 6.20: The differences in SUS Score ratings between the prototype and IntelliJ IDEA per participant.

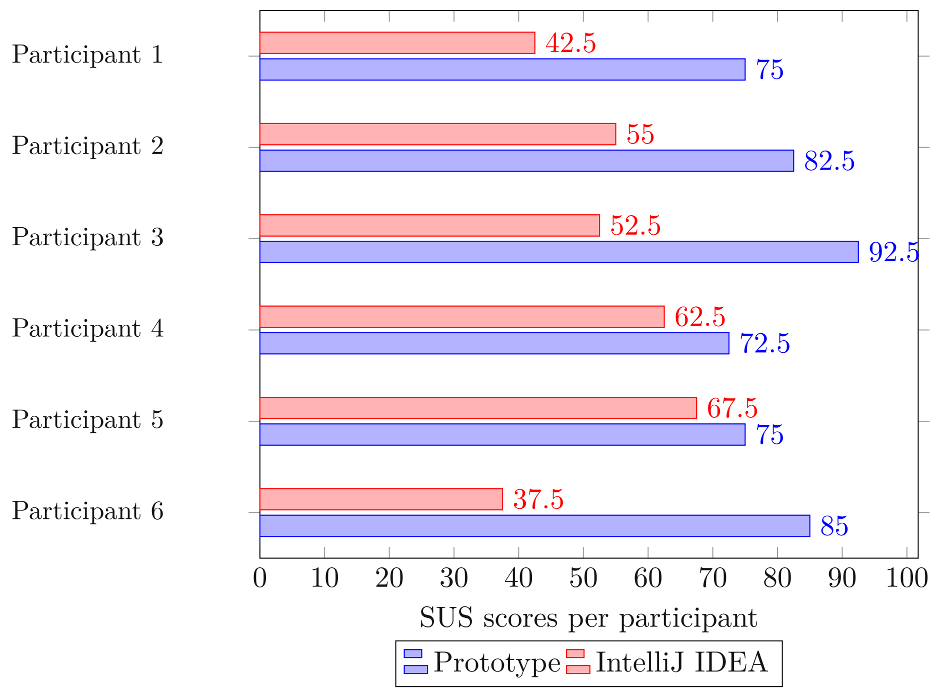


Figure 6.9: SUS scores per participant for both tools.

By utilizing this system and taking the mean of the SUS scores of all participants, we are able to calculate a scoring of 80.42 for the evaluated prototype (standard deviation of 7.65; 95% confidence interval $[72.77, 88.07]$) The calculated score for IntelliJ IDEA resulted in 52.92 (standard deviation of 11.45; 95% confidence interval $[41.47, 64.37]$).

In order to determine the statistical significance when comparing these means, we perform a paired t-test as laid out in [69].

The mean of the difference SUS scores (refer to Table 6.20) between the two candidates amounts to $\hat{D} = 27.5$ with a standard deviation of $S_D = 16.05$. We can calculate the test statistic t the following way: $t = \frac{\frac{\hat{D}}{S_D}}{\sqrt{n}} = 4.20$

The corresponding p-value calculated as described in Subsection 6.2.1 amounts to $0.0085 < 0.05$. The SUS score 80.42 of the prototype is statistically significantly higher than the one of IntelliJ IDEA, 52.92 [69].

When creating a confidence interval around the mean of differences between the two SUS scores, $\hat{D} = 27.5$, as described in Subsection 6.2.1, we get 27.5 ± 6.87. This means that the actual difference in SUS scores between the prototype and IntelliJ IDEA is between 20.62 and 34.37 with a confidence of 95% [69].

6.2.8 Evaluation of Qualitative Feedback

In this section, we will present the qualitative feedback that was received during the evaluation sessions from the participants. The transcripts were analyzed and categorized into the following categories:

1. Data Visualization
2. User Experience
3. Scaling
4. Navigation
5. Integration

When analyzing the transcribed data gathered during the evaluations, we can see that there was a large amount of feedback for both the prototype and IntelliJ IDEA.

Data Visualization

A topic that was commented on by almost all participants was the ability of the tools to visualize data and present it to the user. Strengths and challenges were identified in both evaluated tools.

Prototype Multiple participants appreciated the visual clarity of the prototype: „Der Vorteil war definitiv, dass es [der Prototyp] grafisch viel ansprechender, viel visueller war.“ (P3), „Also Vorteil ist ganz klar, dass es diese Fragestellungen schön grafisch aufarbeitet und die Sachen, die man braucht, halt auf einen Blick sieht.“ (P2).

Regarding evaluating metrics of a software project, the prototype was also seen as being more effective „Also rein von den Metriken her, was man auslesen kann, ist eigentlich meiner Meinung nach der Prototyp jetzt schon besser, als das was IntelliJ hergibt. “ (P4).

The Author-Mode was also received well: „Also, dass angezeigt wird, über welchen Zeitraum welcher Student Contributions gemacht hat. [. . .] Da wird mit prägnanten Attributen gearbeitet. Alles Irrelevante hat keine Farbe, z.B. Weiß, und das Wichtige ist hervorgehoben. [. . .] Diese ganzen On-Hover-Geschichten – das war cool.“ (P6).

However, some drawbacks were noted as well regarding visualization of the directory structure: „Das ganze Repository sollte als ganzen Ordner sichtbar sein.“ (Individual Student Evaluation 4, P1), „Mich stört es, dass man nicht mehrere Ordner aufeinmal sehen kann.“ (Code Frequency Analysis 2, P3)

IntelliJ IDEA The ability to display the information of other branches was seen as something useful in IntelliJ IDEA: P5: „Also im Prototyp sieht man keine Informationen über die Branches." Interviewer: „Das heißt, das ist ein Vorteil bei IntelliJ findest du, dass man das sieht?" P5: „Ich glaube schon, ja."

Other parts were not seen as favorably. Especially the visualization of quantitative data was missed when working with IntelliJ IDEA. One participant noted that they had to count quantitative data by hand: „[. . .] aber wenn man halt irgendwie die Quantität haben möchte, dann muss man es halt nach meinem Wissen abzählen." (P5).

Identifiying contributor statistics or analysing deeper metrics was also seen as not possible: „Nachteil für IntelliJ würde ich jetzt, also der Hauptnachteil für IntelliJ gegenüber deinem Tool, es ist halt wirklich schwer zum Beispiel so eine Contribution Statistik und so auszulesen bei IntelliJ, weil das geht einfach nicht. [. . .] alles, was dann tiefer geht und wirklich Metriken auslesen soll [. . .] da gibt es nichts." (P4).

A lack of seeing a time-frame was identified as well: „[. . .]man sieht halt nicht quasi den Time-Frame. Zeitspanne." (P6).

User Experience

The general user experience was another area that was addressed when using both the prototype and IntelliJ IDEA.

Prototype Areas for improvement were identified regarding the general user experience of the prototype: „Naja, es ist halt so ein Nachteil, es ist halt eher, wie soll man sagen, starr und vorgegebene Funktionen und über das hinaus gibt's nicht viel." (P1).

Text overlapping file bars or bars going off-screen was also observed: „Bei einem Folder war überlappender Text von Files." (P6), „Wenn jetzt der Balken mehr oder weniger offscreen weitergeht, das zum Sagen, dass das da noch nicht aufhört, sondern da kommt noch was." (P4).

Additionally, there was the wish for path information and the contributor list to stay visible on the screen while scrolling: „Mir ist was aufgefallen. Ich hätte mir so was wie eine fixierte Position gewünscht, so eine Bar. Wo man sieht, wo man gerade ist – dass das oben fixiert bleibt und beim Scrollen sichtbar bleibt. [. . .] So eine Leiste – ich weiß nicht, wie das platzmäßig aussieht – aber dass man im Contribution Mode nicht runterscrollen muss, um Studierende auszuwählen oder abzuwählen." (P6).

The fact that the colors used in the prototype can be difficult to see, especially with a larger number of contributors, was also seen as room for improvement in the prototype: „Farben können halt schnell zum Problem werden." (P1), „Farben sind etwas schwierig, bei mehr Developern wirds noch schwieriger." (Refactor Analysis 3, P1).

IntelliJ IDEA When completing the tasks using IntelliJ IDEA, participants noted a few positive aspects of their experience with the tool. They appreciated the ability to use shortcuts: „Du hast dort Shortcuts und so. Es ist halt ganz anders aufgebaut. Das ist der große Vorteil.“ (P3). Other features such as the ability to show the history of a file by right clicking, the commit tree and the included dark mode: „Ich kann auf jeden Fall klicken und dann Rechtsklick machen und Show History. [. . .] Der Commit Tree. [. . .] Es gibt einen Dark Mode.“ (P6).

One identified disadvantage of IntelliJ IDEA was that it was overly complex and features are not so easy to find: „Ein bisschen übermäßig komplex und ich weiß nicht, ob die Features einfach nicht da sind oder ob ich einfach so dumm war, sie zu finden.“ (P2).

The slow startup times of IntelliJ IDEA were also something mentioned during the evaluation: „Also ein Nachteil ist sicher, wenn man wirklich nur schnell was nachschauen will. [. . .] dauert IntelliJ IDEA zu lange zu starten, dass man ans Ziel kommt, ja.“ (P3).

Navigation

Participants commented on the navigation abilities of both tools. Filtering, searching and sorting were features that were missed or seen as incomplete in the prototype, while IntelliJ IDEA was praised for it’s powerful filtering abilities.

Prototype Navigating the prototype when completing the tasks was seen as difficult by a few participants. One participant noted that they would have liked to be able to select multiple folders and filter by them („Was vielleicht auch noch cool wäre, wenn man da mit so einer Checkbox mehrere Ordner anklicken kann. [. . .] Dann kann ich sagen, ich möchte jetzt für die beiden Ordner die gemeinsamen Contribution, so muss ich nicht händisch zusammenzählen.“) (P3).

Filtering authors showed to be confusing when switching to different modes afterwards: „Ich glaube, ich habe gefiltert gehabt und dann war ich in einer anderen Ansicht und habe vergessen, dass ich gefiltert habe und dann hat es eben nur noch die eine Person angezeigt und ich habe mich gewundert, warum der eine war ohne Commits, also ohne Contributions.“ (P5).

Another observed improvement opportunity was a sorting functionality. One participant would have liked to have icons next to the files and directories or to be able to sort them: „Dass man die Dateien und die Folders, dass man da irgendein Icon dazu macht oder die sortiert.“ (P2). While working on the task *Code Frequency Analysis 2*, another participant commented that it was rather hard to do without a sorting ability: „Ohne Sortierung schwierig.“ (Code Frequency Analysis 2, P1).

Next to filtering and sorting, the need to be able to search within the prototype visualization was stated by multiple participants. „Ich hätte mir eine Suche gewünscht, wo ich einfach einen File eingeben kann.“ (P6), „Zum Beispiel ein Suchfeld wäre ganz cool.“ (P2), „Ja, die Suche nach Files [fehlt] auf jeden Fall.“ (P6). While completing the task *Refactor Analysis 1* using the prototype, a participant explicitly stated the wish for a search bar: „Suchleiste wäre cool.“ (Refactor Analysis 1, P4).

IntelliJ IDEA Navigating IntelliJ IDEA, especially the filtering abilities, was commented on positively by a few participants: „Der Vorteil von IntelliJ ist ganz klar, dass diese Filter unglaublich mächtig sind. Kann man sie zwar auch vertun, aber bietet halt einfach mehr.“ (P3), „Es gibt Filteroptionen. Also ich kann nach den Contributors filtern.“ (P6).

Scaling

Prototype The topic of scaling to a larger number of contributors was also addressed. One participant noted that „Die wenigsten Studierendenprojekte haben wirklich nur 6 Contributors.“ (P1).

IntelliJ IDEA Scaling was also addressed by one participant while working on the task *Refactor Analysis 3*: „Nur lösbar weils kein großes Projekt ist, ansonsten wärs mühsam auf diese Weise.“ (Refactor Analysis 3, P4).

Integration

Prototype One downside of the prototype was the fact that it is not integrated in another tool. Participants were noting that they would have to handle yet another tool next to their IDE: „[...] ich brauche wieder ein extra Feature, brauche einen Browser irgendwo man muss sich darum kümmern, dass das aktuell ist.“ (P3), „Es ist halt nochmal ein Umweg quasi, was man gehen muss, was realistisch gesehen, viele Entwickler nicht gehen werden, wenn es nicht in IntelliJ integriert ist, dann muss man schon jemanden haben, der das wirklich wissen will.“ (P4).

IntelliJ IDEA The fact that the features offered by IntelliJ IDEA are already integrated into a widely used IDE was seen as an advantage over the prototype: „Naja, also, der Hauptvorteil für IntelliJ ist relativ offensichtlich, würde ich sagen, wenn man IntelliJ benutzt, hat man die Tools schon. [...] Man braucht halt kein externes Tool.“ (P4)

CHAPTER 7
Discussion

This chapter discusses the research objectives of the thesis and the overall results presented in the previous chapter. Furthermore, we will highlight any threats to the validity of this academic work. Finally, potential future research that could build on top of the presented work is explored.

7.1 Research Objectives

The research questions that this thesis attempted to answer were the following:

7.1.1 RQ1: What are different methods to visualize the evolution of a software repository?

During the first phase of the thesis, research was done to assess existing visualization methods and develop mockups consisting of Storyline, Treemap and Radial Tree visualizations for displaying the history of software repositories. One of these developed prototype mockups, the storyline visualization presented in Subsection 4.1.1 was then chosen for further evaluation.

7.1.2 RQ2: How do domain experts agree with the chosen visualization approach?

The second phase consisted of conducting interviews with three experts with the goal of assessing and gathering feedback on mockups of the previously chosen visualization method. As presented in Section 4.3, the evaluation resulted in an overall very positive impression among all participants. Based on this, we saw that the consulted experts do agree on the usefulness of the proposed visualization method.

7.1.3 RQ3: How does the visualization compare to the state-of-the-art with regard to the following benefits: student project transparency, refactoring frequency, stale code detection, code hotspots?

In order to answer the third research question, another evaluation was conducted where the implemented prototype (see Chapter 5) of the chosen Storyline visualization method was compared to IntelliJ IDEA in a within-subject mixed-method user study. Six participants were interviewed and tasked with completing 13 tasks in four scenarios, which are defined in Subsection 6.1.5 and mapped to the aforementioned benefits as described in Subsection 6.1.9. The results of these evaluations were then interpreted. Based on these insights, the research question was addressed in the following section, specifically in Subsection 7.2.5.

7.2 Result Interpretation

In order to answer **RQ3** described in Subsection 7.1.3, this section analyzes and interprets the results that were found during the evaluation of the developed prototype and are presented in Section 6.2 for each of the proposed benefits.

7.2.1 Project Transparency

For the *Project Transparency* benefit (tasks mapped in Table 6.2), participants achieved a higher mean success rate with the prototype ($\approx 81.25\,\%$) than with IntelliJ IDEA ($\approx 68.75\,\%$), which can be seen in Table 6.6. This difference did not reach statistical significance ($t(5) \approx 1.369, p \approx 0.229 > 0.05$; Cohen's $d_z \approx 0.559$).

The participants perceived *Project Transparency*-related tasks as significantly easier using the prototype with a mean difficulty of ≈ 1.46 vs. ≈ 2.44 ($t(5) = -3.173, p = 0.025 < 0.05; d = -1.295$).

Completion times were similarly significantly shorter (mean $\approx$ 37.65s vs. $\approx$ 62.68s; $t(5) \approx -3.405, p \approx 0.019 < 0.05; d \approx -1.390$).

The qualitative feedback that was given supported the findings described above: the prototype's graphical overview of student contributions was received well („Also Vorteil ist ganz klar, dass es diese Fragestellungen schön grafisch aufarbeitet und die Sachen, die man braucht, halt auf einen Blick sieht.“ - P2) as was the Author-Mode. For the latter, they also noted that any non-relevant information was de-emphasized effectively, making it easy to spot who contributed when. At the same time, feedback also included requests for improved navigation functionality such as fixed UI element containing the current path or a multi-folder filter in order to reduce scrolling when selecting authors or directories.

Overall, the visualization significantly enhanced the speed and ease of understanding project-wide contributions, even if task success rates did not reach statistical significance at $p < 0.05$.

7.2.2 Refactoring Frequency

In contrast, for *Refactoring Frequency* tasks, IntelliJ IDEA did better than the prototype: the success rates with the prototype were only $\approx 33.33\%$ versus $\approx 66.67\%$ with IntelliJ IDEA ($t(5) \approx -3.162, p \approx 0.025 < 0.05; d \approx -1.29$).

With regard to perceived difficulty, tasks were seen as non-significantly easier using the prototype with a mean difficulty of ≈ 2.61 vs. ≈ 2.83 ($t(5) \approx -0.386, p \approx 0.715 > 0.05; d \approx -0.158$).

The task completion times were shorter with IntelliJ IDEA (≈ 50.583s) when compared to the prototype (≈ 72.167s), which was not found to be statistically significant: ($t(5) \approx 2.119, p \approx 0.088 > 0.05; d \approx 0.865$).

Participants struggled to identify refactoring frequencies when using the prototype, commenting that there is limited exploration („Naja, es ist halt so ein Nachteil, es ist halt eher, wie soll man sagen, starr und vorgegebene Funktionen und über das hinaus gibt's nicht viel.“ - P1) and that color distinctions became confusing with multiple contributors. By contrast, IntelliJ IDEA's built-in refactoring indicators, keyboard shortcuts and integrated history views made it easier to locate and count refactoring events.

As a result, the prototype could improve in offering refactoring-specific insights in comparison to IntelliJ IDEA's feature set.

7.2.3 Code Hotspots & Stale Code Detection

For *Code Hotspots & Stale Code Detection*, the prototype showed a higher mean success rate ($\approx 66.67\%$ vs. 50.00% with IntelliJ IDEA), but this did not achieve statistical significance ($t(5) \approx 1.581, p \approx 0.175 > 0.05; d \approx 0.65$).

The mean perceived difficulty was comparable across tools (2.75 for the prototype and ≈ 3.08 for IntelliJ IDEA), and did not reach statistical significance ($t(5) \approx -0.555, p \approx 0.603 > 0.05; d \approx -0.226$).

Participants completed tasks slightly faster using IntelliJ IDEA when compared to the prototype (≈ 84.17s vs. ≈ 94.67s). On average the difference was not significant ($t(2) \approx 2.96, p \approx 0.098 > 0.05; d \approx 1.709$). Here, only three differences were considered for the comparison (participants 2, 4 and 6), as the other participants did not complete any of the tasks for either the prototype or IntelliJ IDEA within the timeout of two minutes.

In sum, it was not possible to determine a significant difference between the tools in this regard, also due to the small size of the dataset.

7.2.4 Usability

The general usability of the prototype in the form of a SUS score was perceived as statistically significantly higher than with IntelliJ IDEA. The prototype scored ≈ 80.42 vs. ≈ 52.92 for IntelliJ IDEA ($p \approx 0.0085 < 0.05, d \approx 1.71$).

7.2.5 Answering RQ3

In summary, both tools performed better in different aspects. The developed prototype was able to do better than IntelliJ IDEA in some of the proposed benefits.

The *Project Transparency* benefit was where this was evident the most. The prototype performed better across all three metrics for that benefit, with the mean perceived difficulty across participants being even statistically significantly shorter. The SUS scoring of the prototype also statistically significantly exceeded the one of IntelliJ IDEA, indicating an advantage in general usability as well.

In the *Refactoring Frequency* metric, the prototype showed disadvantages when compared to IntelliJ IDEA. The results of the metric indicated that the related tasks were statistically significantly more frequently completed successfully with IntelliJ IDEA. They were also non statistically significantly perceived as being easier and faster to complete with the tool.

The *Stale Code Detection & Code Hotspots* benefits show a more mixed story where both tools performed roughly the same, without showing any statistically significant differences.

With these result interpretations in mind, the research question **RQ3** can be answered in the following way: The visualization showed to be partially more useful when compared to existing frameworks. Concretely, the *Project Transparency* benefit and its general usability seem to outperform IntelliJ IDEA, while the results for the remaining benefits indicate either less or similar usefulness.

7.2.6 Practical Implications in Software Engineering Education

The prototype visualization supported participants in assessing team contributions by providing them with an overview across a timeline. In the within-subject mixed-method evaluation presented in Section 6.1, users were able to determine which student authored a given change ≈40% faster with the prototype (mean completion = 37.65s vs. 62.68s; $t(5) = 3.405, p = 0.019$) (see Table 6.16). The qualitative feedback supports this advantage with regard to efficiency in task completion: „Also Vorteil ist ganz klar, dass es diese Fragestellungen schön grafisch aufarbeitet und die Sachen, die man braucht, halt auf einen Blick sieht.“ (P2) (refer to Subsection 6.2.8).

Student Project Analysis

For practice, this means that university assistants or tutors could utilize the storyline-based visualization when evaluating group projects developed in software engineering courses. This way, they could be able to comprehend the progress of the project in a shorter amount of time.

Additionally, students could also use the visualization to get a better feeling for how their software project is evolving and to make them aware of the collaboration within the team, increasing motivation and engagement, which is also suggested by [60].

Grading

The visualization could also assist in the grading process performed by university assistants and tutors by helping them interpret the contributions of students to a group project. Especially using the author-mode (see Section 4.1.1), individual student contributions can be highlighted and provide a more objective basis for grading. For instance, the significantly lower perceived difficulty (mean difficulty = 1.46 vs. 2.44; $t(5) = 3.173, p = 0.025$) when using the prototype for such tasks suggests that educators could make fewer errors in their evaluations of students (see Subsection 6.2.5).

7.3 Threats to Validity

There are some factors that have to be considered when looking at the results presented in this thesis, which could undermine its reliability and accuracy.

7.3.1 Small sample size

Due to resource and time constraints, only six participants could be interviewed in the final evaluation. This makes the presented results possibly inaccurate, which can be seen in Section 6.2 where most of the differences between the prototype and IntelliJ IDEA regarding the proposed metrics could not be shown to be statistically significant.

7.3.2 Commits as Contributions

The way the prototype presents contributions was based on commit data. This may not be the most effective way to determine contributions of students. Still, the purpose of the evaluation was to explore the visualization idea in the form of this prototype.

7.3.3 Flaws in Prototype

The implementation of the visualization idea was realized in the form of a prototype due to the limited scope of this thesis. There were flaws in the application that could have influenced the results of the evaluation, as in some cases a feature was not working correctly (see Subsection 5.4.5 and Subsection 5.3.3, which discuss this in more detail).

Further development will be necessary, or even a re-implementation of the idea to create a more mature version of this visualization idea.

7.3.4 Time Limit for Task Completion

During the pilot evaluation sessions, the time limit of 120 seconds for each task was deemed to be sufficient for the real evaluations. This limit was chosen based on the time the tasks took to complete in the pilot sessions. Also, it was recognized that the following prototype evaluation sessions would have to be time-boxed in order to not take too long, preventing an impairment of the concentration of the candidates, as discussed in Subsection 6.1.1.

Still, this time limit caused the time of the candidates to be cut short during the work on tasks in some cases during the prototype evaluations. This could mean that task results for the prototype or IntelliJ IDEA were influenced by stress caused by the time limit or there not being enough time to complete a task for the candidates.

7.3.5 Bias of Interviewer

The person that moderated both evaluation sessions was the same person that authored this thesis. While they tried their best to stay neutral during the evaluation processes, it may not always have been possible to eliminate a certain bias in the way they reacted to answers in a non-verbal way.

7.3.6 Construct Validity

There was no way of knowing whether the tested scenarios effectively measured benefits such as *Project Transparency*, *Refactoring Frequency* or *Stale Code Detection & Code Hotspots*.

One attempt to tackle this threat to validity was to gather qualitative feedback where the participants were able to express their thoughts on the evaluated tools verbally (see Subsection 6.1.8).

7.3.7 Task Mapping

As already mentioned in Subsection 6.1.9, the majority of tasks is mapped to the *Project Transparency* benefit. This caused less focus being put on the *Refactoring Frequency* and *Stale Code Detection & Code Hotspots* benefits. The reason for that was the identified importance that such visualizations play in comprehension of student projects for educators, especially with regard to the analysis of individual student contributions [48] [70] [25] [50], and considering that the focus of this thesis was on the educational aspects of software project visualization.

7.4 Ethical Considerations

Ensuring that ethical academic practices were adhered to was of high importance when working on this thesis. In order to achieve that, the following measures were implemented.

7.4.1 Informed Consent

Before collecting data from participants, they were provided with a written information where the purpose of the evaluation, what the data would be used for and that they could withdraw at any time.

7.4.2 Privacy and Confidentiality

All personally identifiable data gathered during the evaluations, such as names, were replaced with unique codes such as „P1“ immediately upon collection. Audio recordings of the evaluation sessions were only shared via secure file-sharing services, concretely RISE File Drop [91] and removed after their analysis and anonymized transcription.

7.4.3 Data Anonymization and Pseudonymization

The data in the used testing repository underwent an anonymization process. Names and email addresses were replaced with anonymized identifiers (e.g. „student1“ or „student1@example.org“).

7.5 Future Work

The goal of this thesis was to evaluate the usefulness of the visualization idea, which future work can build upon. We developed a visualization prototype to determine whether it is more useful with regard to the proposed benefits than IntelliJ IDEA. The results were indicators towards higher usefulness, effectiveness and efficiency by the prototype with regard to the *Project Transparency* benefit, which the thesis put the most emphasis on. The other benefits showed mixed results, with *Refactoring Frequency* showing more advantages on the side of IntelliJ IDEA, and *Stale Code Detection & Code Hotspots* having mixed results. Most of these results were not statistically significant, which can be attributed to the small sample size of six participants. Based on the findings presented in this thesis, the following subsections propose potential future contributions to our visualization approach.

7.5.1 Additional Evaluations

Our evaluations showed indicators of the usefulness of the proposed visualization idea. Additional studies will be needed to confirm these mostly non-statistically significant findings with mostly large effect sizes. Larger pools of candidates will be needed, which will require additional time and resources. The design of future evaluations could be adjusted in a way such that the time limit for task completion is further evaluated, possibly increasing it by a small amount of time to get an even clearer picture on the usefulness of the visualization idea. More scenarios and tasks could be added in order to also focus more on the benefits next to *Project Transparency* in an extended task-to-benefit mapping.

7.5.2 Integration Into Other Tools

One potential future area for exploration could be how the visualization can be integrated into existing IDEs or tools in order to address the feedback regarding a lack of integration (see Subsection 6.2.8). As presented in Subsection 3.1.8, one candidate for this could be the *Git-Truck* project [33], which focuses on a similar approach as our visualization idea. Perhaps, future work could even go as far as contributing our storyline-based visualization prototype to this open-source project [87], in order to enhance it further and utilize its already mature architecture. This would make it possible to perform future evaluations of our visualization approach as part of this production-ready tool that could then also be utilized in an educational context.

7.5.3 Additional Features

One idea for the prototype worth exploring further that came up during an evaluation was to add a Git graph query implementation in order to explore the history of a software project even further. Also, as mentioned in Subsection 3.2.3, Subsection 3.2.4 and Subsection 3.2.7, there are areas in educational software engineering research that can be considered for enhancing the contribution analysis features of our proposed visualization.

CHAPTER 8

Conclusion

This thesis attempted to find out whether a storyline-based visualization of the directory and file evolution in software repositories can improve the comprehension of student projects in an educational setting. To address this, three research questions were defined:

- **RQ1:** What are different methods to visualize the evolution of a software repository?
- **RQ2:** How do domain experts agree with the chosen visualization approach?
- **RQ3:** How does the visualization compare to the state-of-the-art with regard to the following benefits: student project transparency, refactoring frequency, stale code detection, code hotspots?

For **RQ1**, an overview of relevant visualization techniques was provided, and three prototypes were designed. The storyline visualization was chosen for further evaluation as it was able to display operations across an entire timeline and highlight individual contributions. For **RQ2**, interviews with professionals in software engineering and education were performed. These confirmed the perceived usefulness and applicability of the chosen approach. For **RQ3**, a mixed-method user study compared the implemented prototype with IntelliJ IDEA, showing significant advantages in *Project Transparency*-related tasks, higher SUS scores, and positive qualitative feedback, while IntelliJ IDEA outperformed the prototype in the tasks targeting *Refactoring Frequency* and both tools performed similarly for *Stale Code Detection & Code Hotspots*.

The results presented in this thesis suggest that storyline-based visualizations could improve the efficiency and usefulness to understand student project contributions, especially in the context of software engineering education. While existing IDE tools, namely IntelliJ IDEA, still perform better in other areas such as *Refactoring Analysis*, integrating such storyline-based visualizations into educational workflows could have the potential to improve student project transparency, support educators in achieving a fairer grading process, and to achieve a deeper understanding of software project evolution. Future work could focus on further evaluations with larger sample sizes, integration of the storyline-based visualization into existing IDEs and tools, and expanding the range of tasks with regard to the other benefits.

List of Figures

List of Tables

Acronyms

AST Abstract Syntax Trees. 8, 9, 18, 45, 47

CI Continuous Integration. 7

CLI Command-Line Interface. 60, 67, 68

CVCS Centralized Version Control Systems. 5, 6

CVS Concurrent Versions System. 5

DAG Directed Acyclic Graph. 7

DOM Document Object Model. 56

DVCS Distributed Version Control Systems. 2, 6, 7

IDE Integrated Development Environment. 13, 29, 59, 87, 97, 98

LOC Lines Of Code. 14, 17

MSR Mining Software Repositories. 1, 3, 7–9, 98

NPM Node Package Manager. 58

OSS Open Source Software. 6

SCCS Source Code Control Systems. 5, 8

SUS System Usability Scale. 64, 82, 83, 92, 97

SVG Scalable Vector Graphics. 56

SVN Apache Subversion. 5, 8

TDD Test-Driven Development. 21

UI User Interface. 57, 58

URL Uniform Resource Locator. 45

UX User Experience. 57, 58

VCS Version Control Systems. 3, 5, 6, 29

Bibliography

[1] E Aghajani, A Mocci, G Bavota, and M Lanza. The Code Time Machine. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 356–359, 2017. doi: 10.1109/ICPC.2017.6.

[2] Nour Jihene Agouf, Soufyane Labsari, Stéphane Ducasse, Anne Etien, and Nicolas Anquetil. A Visualization for Client-Server Architecture Assessment. page 255. Université de Limoges, Université de Limoges, 10 2023. doi: 10.34894/VQ1DJA. URL `https://hal.science/hal-04231797https://hal.science/hal-04231797/document`.

[3] Wolfgang Aigner, Silvia Miksch, H Schumann, and Christian Tominski. A Survey of Visualization Techniques - Storyline Visualization. In *Visualization of Time-Oriented Data*, chapter Back Matter, pages 264–264. Springer London, 9 2023. ISBN 978-1-4471-7526-1. doi: 10.1007/978-1-4471-7527-8.

[4] Wolfgang Aigner, Silvia Miksch, H Schumann, and Christian Tominski. Crafting Visualizations of Time-Oriented Data - Static vs. Dynamic Time-Series Visualizations. In *Visualization of Time-Oriented Data*, pages 93–94. Springer London, London, 2023. ISBN 978-1-4471-7527-8. doi: 10.1007/978-1-4471-7527-8{_}4. URL `https://doi.org/10.1007/978-1-4471-7527-8_4`.

[5] Danielle Albers, Michael Correll, and Michael Gleicher. Task-driven evaluation of aggregation in time series visualization. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 551–560. Association for Computing Machinery, 2014. ISBN 9781450324731. doi: 10.1145/2556288.2557200. URL `https://dl.acm.org/doi/10.1145/2556288.2557200`.

[6] Diane Lindwarm Alonso, Anne Rose, Catherine Plaisant, and Kent L. Norman. Viewing personal history records: A comparison of tabular format and graphical presentation using LifeLines. *Behaviour & Information Technology*, 17(5):249–262, 1 1998. ISSN 13623001. doi: 10.1080/014492998119328. URL `https://www.tandfonline.com/doi/abs/10.1080/014492998119328`.

[7] Aaron Bangor, Philip T Kortum, and James T Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction*,

24(6):574–594, 2008. doi: 10.1080/10447310802205776. URL `https://doi.org/10.1080/10447310802205776`.

[8] Laure Bedu, Olivier Tinh, and Fabio Petrillo. A tertiary systematic literature review on software visualization. In *Proceedings - 7th IEEE Working Conference on Software Visualization, VISSOFT 2019*, pages 33–44. Institute of Electrical and Electronics Engineers Inc., 9 2019. ISBN 9781728149394. doi: 10.1109/VISSOFT.2019.00013.

[9] Dirk Beyer. Co-change visualization applied to PostgreSQL and ArgoUML: (MSR challenge report). In *Proceedings - International Conference on Software Engineering*, pages 165–166, 2006. ISBN 1595933972. doi: 10.1145/1137983.1138023. URL `https://dl.acm.org/doi/10.1145/1137983.1138023`.

[10] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. The promises and perils of mining git. *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, pages 1–10, 2009. doi: 10.1109/MSR.2009.5069475.

[11] John Brooke and others. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[12] Ruven Brooks. Towards a theory of the cognitive processes in computer programming. *International Journal of Human-Computer Studies*, 51(2):197–211, 8 1977. ISSN 1071-5819. doi: 10.1006/ijhc.1977.0306.

[13] Scott Chacon and Ben Straub. *Pro Git (Second Edition)*. Springer Nature, 2014. ISBN 9781484200766. doi: 10.1007/978-1-4842-0076-6.

[14] Gary Charness, Uri Gneezy, and Michael A. Kuhn. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization*, 81(1):1–8, 1 2012. ISSN 0167-2681. doi: 10.1016/J.JEBO.2011.08.009.

[15] Timothy Clem and Patrick Thomson. Static analysis at GitHub. *Commun. ACM*, 65(2):44–51, 1 2022. ISSN 0001-0782. doi: 10.1145/3486594. URL `https://doi.org/10.1145/3486594`.

[16] Marco D'Ambros and Michele Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, pages 229–238, 2006. ISBN 0769525369. doi: 10.1109/CSMR.2006.51.

[17] Brian De Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering, CHASE 2009*, pages 36–39, 2009. doi: 10.1109/CHASE.2009.5071408.

[18] Santiago Perez De Rosso and Daniel Jackson. Purposes, concepts, misfits, and a redesign of git. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pages 292–310, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450344449. doi: 10.1145/2983990.2984018. URL `https://doi.org/10.1145/2983990.2984018`.

[19] Eduardo Faccin Vernier, Alexandru C. Telea, and Joao Comba. Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization. In *Proceedings - 6th IEEE Working Conference on Software Visualization, VISSOFT 2018*, pages 96–106. Institute of Electrical and Electronics Engineers Inc., 11 2018. ISBN 9781538682920. doi: 10.1109/VISSOFT.2018.00018.

[20] Johannes Feiner and Keith Andrews. RepoVis: Visual Overviews and Full-Text Search in Software Repositories. In *Proceedings - 6th IEEE Working Conference on Software Visualization, VISSOFT 2018*, pages 1–11. Institute of Electrical and Electronics Engineers Inc., 11 2018. ISBN 9781538682920. doi: 10.1109/VISSOFT.2018.00009.

[21] A Fujimoto, Y Higo, and S Kusumoto. Towards Accurate File Tracking Based on AST Differences. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, pages 553–558, 2021. ISBN 2640-0715. doi: 10.1109/APSEC53868.2021.00067.

[22] H Gall, M Jazayeri, and C Riva. Visualizing software release histories: the use of color and third dimension. In *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, pages 99–108, 1999. doi: 10.1109/ICSM.1999.792584.

[23] Rose F. Gamble and Matthew L. Hale. Assessing individual performance in Agile undergraduate software engineering teams. *Proceedings - Frontiers in Education Conference, FIE*, pages 1678–1684, 2013. ISSN 15394565. doi: 10.1109/FIE.2013.6685123.

[24] Michael Guttmann, Aleksandar Karakas, and Denis Helic. Attribution of Work in Programming Teams with Git Reporter. *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, 1:436–442, 3 2024. doi: 10.1145/3626252.3630785. URL `https://tugraz.elsevierpure.com/de/publications/attribution-of-work-in-programming-teams-with-git-reporter`.

[25] Sivana Hamer, Christian Quesada-López, Alexandra Martínez, and Marcelo Jenkins. Measuring students' contributions in software development projects using Git metrics. In *2020 XLVI Latin American Computing Conference (CLEI)*, pages 531–540, 2020. doi: 10.1109/CLEI52000.2020.00068.

[26] Sivana Hamer, Christian Quesada-López, Alexandra Martínez, and Marcelo Jenkins. Measuring Students' Source Code Quality in Software Development Projects Through Commit-Impact Analysis. In Álvaro Rocha, Carlos Ferrás, Paulo Carlos López-López, and Teresa Guarda, editors, *Information Technology and Systems*, pages 100–109, Cham, 2021. Springer International Publishing. ISBN 978-3-030-68418-1.

[27] Sivana Hamer, Christian Quesada-Lopez, and Marcelo Jenkins. Students' perceptions of integrating a contribution measurement tool in software engineering projects. *Software Engineering Education Conference, Proceedings*, 2023-August:21–30, 2023. ISSN 10930175. doi: 10.1109/CSEET58097.2023.00013.

[28] Ahmed E Hassan. The road ahead for mining software repositories. In *Proceedings of the 2008 Frontiers of Software Maintenance, FoSM 2008*, page 48 – 57, 2008. doi: 10.1109/FOSM.2008.4659248. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-57849119318&doi=10.1109%2fFOSM.2008.4659248&partnerID=40&md5=126469082e23952269fcb5a98cd4bcb4`.

[29] Jane Huffman Hayes, Timothy C. Lethbridge, and Daniel Port. Evaluating individual contribution toward group software engineering projects. *Proceedings - International Conference on Software Engineering*, pages 622–627, 2003. ISSN 02705257. doi: 10.1109/ICSE.2003.1201246.

[30] Nicole Herbert. Quantitative Peer Assessment: Can students be objective? 2007. doi: 10.5555/1273672.1273680. URL `https://dl.acm.org/doi/10.5555/1273672.1273680`.

[31] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE '16, pages 426–437, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450338455. doi: 10.1145/2970276.2970358. URL `https://doi.org/10.1145/2970276.2970358`.

[32] Adrian Hoff, Thomas Hoffmann Kilbak, Leonel Merino, and Mircea Lungu. Git-Truck@Duck - Interactive Time Range Selection in Hierarchy-Oriented Polymetric Visualization of Git Repository Evolution. *Proceedings - 2024 IEEE International Conference on Software Maintenance and Evolution, ICSME 2024*, pages 853–857, 2024. doi: 10.1109/ICSME58944.2024.00090.

[33] K. Hojelse, T. Kilbak, J. Rossum, E. Japelt, L. Merino, and M. Lungu. Git-Truck: Hierarchy-Oriented Visualization of Git Repository Evolution. *Proceedings - 2022 Working Conference on Software Visualization, VISSOFT 2022*, pages 131–140, 2022. doi: 10.1109/VISSOFT55257.2022.00021.

[34] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, Inc, 2010. ISBN 978-0-321-60191-9.

[35] C. D. Hundhausen, P. T. Conrad, A. S. Carter, and O. Adesope. Assessing individual contributions to software engineering projects: a replication study. *Computer Science Education*, 32(3):335–354, 7 2022. ISSN 17445175. doi: 10.1080/08993408.2022.2071543. URL `https://www.tandfonline.com/doi/abs/10.1080/08993408.2022.2071543`.

[36] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 6 2002. ISSN 1045-926X. doi: 10.1006/JVLC.2002.0237.

[37] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd Conference on Visualization 1991, VIS 1991*, pages 284–291. Association for Computing Machinery, Inc, 10 1991. ISBN 0818622458. doi: 10.1109/VISUAL.1991.175815.

[38] Patrick W Jordan, Bruce Thomas, Ian Lyall McClelland, and Bernard Weerdmeester. *Usability evaluation in industry*. CRC press, 1996.

[39] Woosung Jung, Eunjoo Lee, and Chisu Wu. A Survey on Mining Software Repositories. *IEICE Trans. Inf. Syst.*, E95-D(5):1384–1406, 2012. ISSN 17451361. doi: 10.1587/TRANSINF.E95.D.1384.

[40] Judy Kay, Irena Koprinska, and Kalina Yacef. Educational Data Mining to Support Group Work in Software Development Projects. In *Handbook of Educational Data Mining*. CRC Press, 10 2010. doi: 10.1201/b10274-15.

[41] Nasraldeen Khleel and Károly Nehéz. Mining Software Repository: an Overview. *Doktoranduszok Fóruma*, 6 2020.

[42] Youngtaek Kim, Jaeyoung Kim, Hyeon Jeon, Young Ho Kim, Hyunjoo Song, Bohyoung Kim, and Jinwook Seo. Githru: Visual analytics for understanding software development history through git metadata analysis. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):656–666, 2 2021. ISSN 19410506. doi: 10.1109/TVCG.2020.3030414.

[43] Falko Koetter, Monika Kochanowski, Maximilien Kintz, Benedikt Kersjes, Ivan Bogicevic, and Stefan Wagner. Assessing software quality of agile student projects by data-mining software repositories. In *CSEDU 2019 - Proceedings of the 11th International Conference on Computer Supported Education*, volume 2, pages 244–251. SciTePress, 2019. ISBN 9789897583674. doi: 10.5220/0007688602440251.

[44] Kamilla Kopec-Harding, Sukru Eraslan, Bowen Cai, Suzanne M. Embury, and Caroline Jay. The impact of unequal contributions in student software engineering team projects. *Journal of Systems and Software*, 206:111839, 12 2023. ISSN 0164-1212. doi: 10.1016/J.JSS.2023.111839.

[45] A Kuhn and M Stocker. CodeTimeline: Storytelling with versioning data. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1333–1336, 2012. doi: 10.1109/ICSE.2012.6227086.

[46] Daniel Lakens. Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs. *Frontiers in Psychology*, 4, 2013. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00863. URL `https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2013.00863`.

[47] Hao Lü and James Fogarty. Cascaded treemaps: examining the visibility and stability of structure in treemaps. In *Proceedings of Graphics Interface 2008*, GI '08, pages 259–266, CAN, 2008. Canadian Information Processing Society. ISBN 9781568814230.

[48] Mircea Lungu, Rolf Helge Pfeiffer, Marco D'Ambros, Michele Lanza, and Jesper Findahl. Can Git Repository Visualization Support Educators in Assessing Group Projects? In *Proceedings - 2022 Working Conference on Software Visualization, VISSOFT 2022*, pages 187–191. Institute of Electrical and Electronics Engineers Inc., 2022. ISBN 9781665480925. doi: 10.1109/VISSOFT55257.2022.00030.

[49] Martin Macak, Daniela Kruzelova, Stanislav Chren, and Barbora Buhnova. Using process mining for Git log analysis of projects in a software development course. *Education and Information Technologies*, 26(5):5939–5969, 9 2021. ISSN 15737608. doi: 10.1007/s10639-021-10564-6.

[50] Ana Milovanović, Danijela Stojanović, and Dušan Barać. Exploring Possibilities of Integrating Version Control Platforms in Higher Education Through GitHub Data Analysis. *JWEE*, 3:113–133, 12 2021. doi: 10.28934/jwee21.34.pp113-133.

[51] Megha Mittal and Ashish Sureka. Process mining software repositories from student projects in an undergraduate software engineering course. In *36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings*, pages 344–353. Association for Computing Machinery, 2014. ISBN 9781450327688. doi: 10.1145/2591062.2591152.

[52] Steven Monteiro, Erikas Sokolovas, Ellen Wittingen, Tom van Dijk, and Marieke Huisman. IntelliJML: a JML plugin for IntelliJ IDEA. In *Proceedings of the 23rd ACM International Workshop on Formal Techniques for Java-like Programs*, FTfJP '21, pages 39–42, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385435. doi: 10.1145/3464971.3468423. URL `https://doi.org/10.1145/3464971.3468423`.

[53] Marcello Henrique Dias De Moura, Hugo Alexandre Dantas Do Nascimento, and Thierson Couto Rosa. Extracting new metrics from version control system for the comparison of software developers. In *Proceedings - 28th Brazilian Symposium on Software Engineering, SBES 2014*, pages 41–50. Institute of Electrical and Electronics Engineers Inc., 10 2014. ISBN 9781479942237. doi: 10.1109/SBES.2014.25.

[54] Ilona Nawrot and Antoine Doucet. Timeline Localization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8510 LNCS(PART 1):611–622, 2014. ISSN 1611-3349. doi: 10.1007/978-3-319-07233-3{_}56. URL `https://link.springer.com/chapter/10.1007/978-3-319-07233-3_56`.

[55] Bao An Nguyen, Kuan Yu Ho, and Hsi Min Chen. Measure Students' Contribution in Web Programming Projects by Exploring Source Code Repository. In *Proceedings - 2020 International Computer Symposium, ICS 2020*, pages 473–478. Institute of Electrical and Electronics Engineers Inc., 12 2020. ISBN 9781728192550. doi: 10.1109/ICS51289.2020.00099.

[56] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. Software evolution visualization: A systematic mapping study. *Information and Software Technology*, 55(11):1860–1883, 11 2013. ISSN 0950-5849. doi: 10.1016/J.INFSOF.2013.05.008.

[57] Shazna Nuzrath, Nimesha Hansani Amarasinghe, Kusheni Tharushika Liyanage, Kushnara Suriyawansa, Dakheela Pyaree Madanayake, and Nuwan Kodagoda. GCodex: A tool to analyze software repositories over time (Visualization). In *2019 International Conference on Advancements in Computing, ICAC 2019*, pages 174–179. Institute of Electrical and Electronics Engineers Inc., 12 2019. ISBN 9781728141701. doi: 10.1109/ICAC49085.2019.9103390.

[58] Michael Ogawa and Kwan-Liu Ma. Software Evolution Storylines. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pages 35–42, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0028-5. doi: 10.1145/1879211.1879219. URL `http://doi.acm.org/10.1145/1879211.1879219`.

[59] Takayuki Omori and Katsuhisa Maruyama. A change-aware development environment by recording editing operations of source code. In *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, pages 31–34, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-024-1. doi: 10.1145/1370750.1370758. URL `http://doi.acm.org/10.1145/1370750.1370758`.

[60] Reza M Parizi, Paola Spoletini, and Amritraj Singh. Measuring Team Members' Contributions in Software Engineering Projects using Git-driven Technology. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2018. doi: 10.1109/FIE.2018.8658983.

[61] Jean-Baptiste Raclet and Franck Silvestre. Git4School: A dashboard for supporting teacher interventions in software engineering courses. In *Addressing Global Challenges and Quality Education: EC-TEL 2020 Proceedings*, volume 12315 of *Lecture Notes in Computer Science*, pages 392–397, Cham, 2020. Springer. ISBN 978-3-030-57717-9. doi: 10.1007/978-3-030-57717-9_33.

[62] V. Rajlich and N. Wilde. The role of concepts in program comprehension. In *Proceedings - IEEE Workshop on Program Comprehension*, volume 2002-January, pages 271–278. IEEE Computer Society, 2002. ISBN 0769514952. doi: 10.1109/WPC.2002.1021348.

[63] Peter C. Rigby, Earl T. Barr, Christian Bird, Prem Devanbu, and Daniel M. German. What effect does Distributed Version Control have on OSS project organization? *2013 1st International Workshop on Release Engineering, RELENG 2013 - Proceedings*, pages 29–32, 2013. doi: 10.1109/RELENG.2013.6607694.

[64] Gregorio Robles and Jesus M Gonzalez-Barahona. Mining student repositories to gain learning analytics. An experience report. In *2013 IEEE Global Engineering Education Conference (EDUCON)*, pages 1249–1254, 2013. doi: 10.1109/EduCon.2013.6530267.

[65] Marc J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, 1975. ISSN 00985589. doi: 10.1109/TSE.1975.6312866.

[66] D Rozenberg, I Beschastnikh, F Kosmale, V Poser, H Becker, M Palyart, and G C Murphy. Comparing Repositories Visually with RepoGrams. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 109–120, 2016. doi: 10.1109/MSR.2016.020. URL `https://doi.org/10.1145/2901739.2901768`.

[67] Yusuke Sakaguchi, Takashi Ishio, Tetsuya Kanda, and Katsuro Inoue. Extracting a unified directory tree to compare similar software products. *2015 IEEE 3rd Working Conference on Software Visualization, VISSOFT 2015 - Proceedings*, pages 165–169, 11 2015. doi: 10.1109/VISSOFT.2015.7332430.

[68] Jeff Sauro and James R Lewis. Chapter 5 - Is There a Statistical Difference between Designs? In Jeff Sauro and James R Lewis, editors, *Quantifying the User Experience*, pages 63–103. Morgan Kaufmann, Boston, 2012. ISBN 978-0-12-384968-7. doi: https://doi.org/10.1016/B978-0-12-384968-7.00005-9. URL `https://www.sciencedirect.com/science/article/pii/B9780123849687000059`.

[69] Jeff Sauro and James R. Lewis. *Quantifying the User Experience: Practical Statistics for User Research, Second Edition.* Elsevier, 1 2016. ISBN 9780128023082. URL `http://www.sciencedirect.com:5070/book/9780128023082/quantifying-the-user-experience`.

[70] María Luisa Sein-Echaluce, Angel Fidalgo-Blanco, Francisco José García-Peñalvo, and David Fonseca. Impact of Transparency in the Teamwork Development through Cloud Computing. *Applied Sciences*, 11(9), 2021. ISSN 2076-3417. doi: 10.3390/app11093887. URL `https://www.mdpi.com/2076-3417/11/9/3887`.

[71] Mojtaba Shahin, Peng Liang, and Muhammad Ali Babar. A systematic review of software architecture visualization techniques. *Journal of Systems and Software*, 94: 161–185, 8 2014. ISSN 0164-1212. doi: 10.1016/J.JSS.2014.03.071.

[72] Tan Tang, Renzhong Li, Xinke Wu, Shuhan Liu, Johannes Knittel, Steffen Koch, Thomas Ertl, Lingyun Yu, Peiran Ren, and Yingcai Wu. PlotThread: Creating Expressive Storyline Visualizations using Reinforcement Learning. *CoRR*, abs/2009.00249, 2020. URL `https://arxiv.org/abs/2009.00249`.

[73] Raga'ad M. Tarawneh, Patric Keller, and Achim Ebert. A general introduction to graph visualization techniques. In *OpenAccess Series in Informatics*, volume 27, pages 151–164, 2012. doi: 10.4230/OASICS.VLUDS.2011.151.

[74] Adam Tutko, Austin Henley, and Audris Mockus. How are Software Repositories Mined? A Systematic Literature Review of Workflows, Methodologies, Reproducibility, and Tools, 9 2022.

[75] A. Von Mayrhauser and A. M. Vans. From program comprehension to tool requirements for an industrial environment. In *IEEE International Conference on Program Comprehension*, pages 78–86. IEEE Computer Society, 1993. ISBN 0818640421. doi: 10.1109/WPC.1993.263903.

[76] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In *VISSOFT 2007 - Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99, 2007. ISBN 1424406005. doi: 10.1109/VISSOF.2007.4290706.

[77] Tao Xie, Thomas Zimmermann, and Arie Van Deursen. Introduction to the special issue on mining software repositories. *Empirical Software Engineering*, 18(6):1043–1046, 12 2013. ISSN 13823256. doi: 10.1007/S10664-013-9273-9/METRICS. URL `https://link.springer.com/article/10.1007/s10664-013-9273-9`.

[78] Yan Yan, Massimiliano Menarini, and William Griswold. Mining software contracts for software evolution. In *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, pages 471–475. Institute of Electrical and Electronics Engineers Inc., 12 2014. ISBN 9780769553030. doi: 10.1109/ICSME.2014.76.

[79] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. Version Control System: A Review. *Procedia Computer Science*, 135:408–415, 1 2018. ISSN 1877-0509. doi: 10.1016/J.PROCS.2018.08.191.

[80] Weiqin Zou, Weiqiang Zhang, Xin Xia, Reid Holmes, and Zhenyu Chen. Branch Use in Practice: A Large-Scale Empirical Study of 2,923 Projects on GitHub. *Proceedings - 19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019*, pages 306–317, 7 2019. doi: 10.1109/QRS.2019.00047.

Web Sources

[81] Angular. URL https://angular.dev.

[82] Introduction - Tree-sitter. URL https://tree-sitter.github.io/tree-sitter/index.html.

[83] isomorphic-git · A pure JavaScript implementation of git for node and browsers! URL https://isomorphic-git.org/.

[84] Java. URL https://dev.java.

[85] React. URL https://react.dev.

[86] TypeScript: JavaScript With Syntax For Types. URL https://www.typescriptlang.org/.

[87] Git Truck, 2025. URL https://github.com/git-truck/git-truck.

[88] Randall Munroe. Movie Narrative Charts, 2009. URL https://xkcd.com/657/.

[89] Observable. What is D3? | D3 by Observable, 2024. URL https://d3js.org/what-is-d3.

[90] Eugen Paraschiv. Java IDEs in 2024 – Survey, 2024. URL https://www.baeldung.com/java-ides-in-2024.

[91] RISE. RISE File Drop, 2025. URL https://filedrop.at.rise-applications.com/about.

[92] Brian Vermeer. JVM ECOSYSTEM REPORT 2021, 2021.

APPENDIX A

Mockup Evaluation Survey

The following survey was used for the evaluation of the mockup described in Section 4.3

Visualizing Directory and File Evolutions for Software Repositories in Software Engineering Education

This survey evaluates the prototype visualization that was developed as part of the diploma thesis "Visualizing Directory and File Evolutions for Software Repositories in Software Engineering Education".

The prototype provides an overview over a software project by visualizing
individual directories and files as storylines.
This way, it is possible to comprehend how, when and who changed which parts of the
project architecture.
This can be utilized in an educational context by e.g. tutors and student assistants
to quickly be able to evaluate a group project with regards to structure, past changes, code frequency, stale code detection,
individual contributions and branch comparison.

* Indicates required question

1. What is your role closest to? *

Mark only one oval.

- ◯ Software Developer
- ◯ Tutor
- ◯ University Assistant
- ◯ Project Manager
- ◯ Other: ______________________

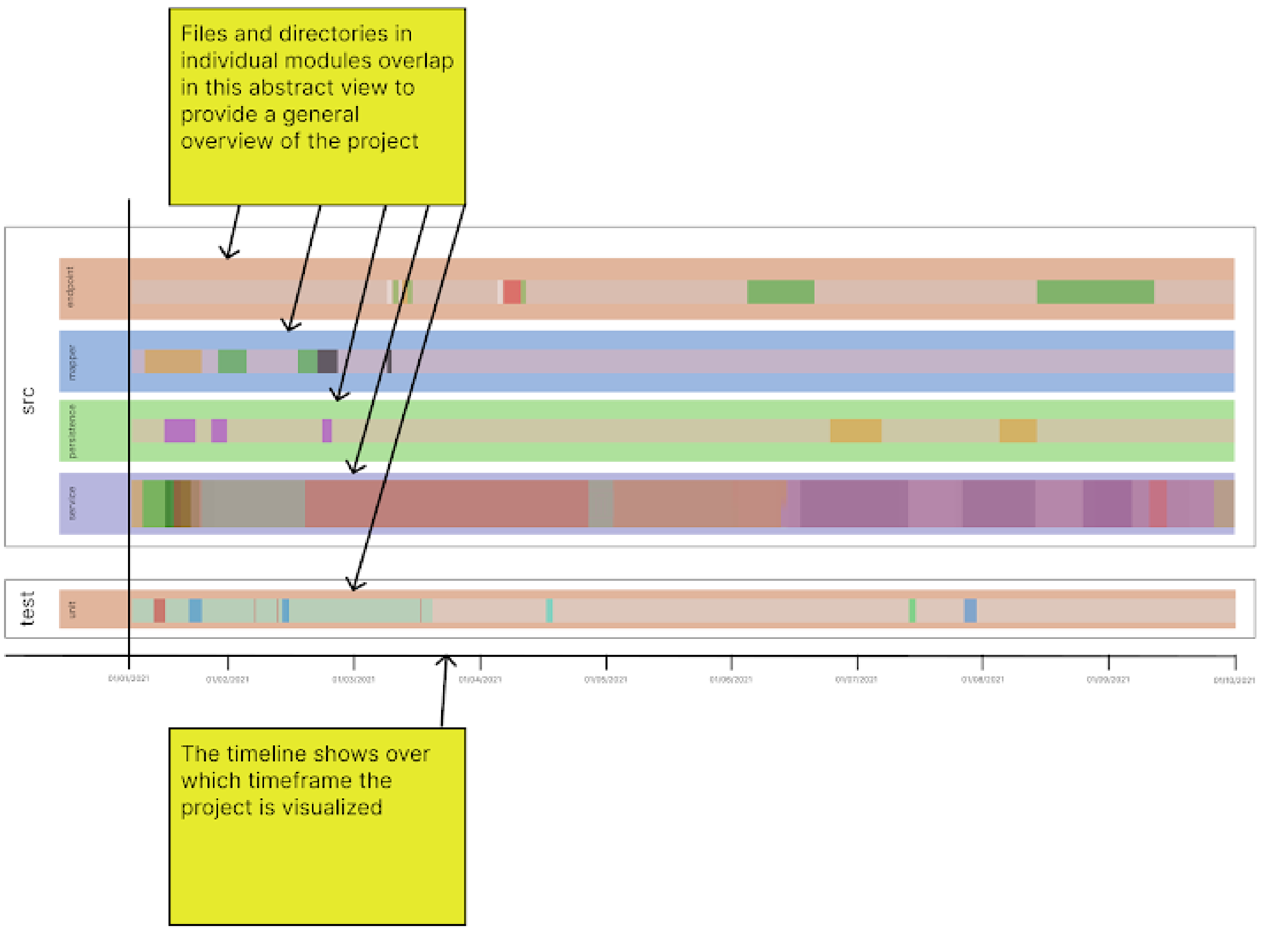
2. How many years of professional experience do you have? *

Mark only one oval.

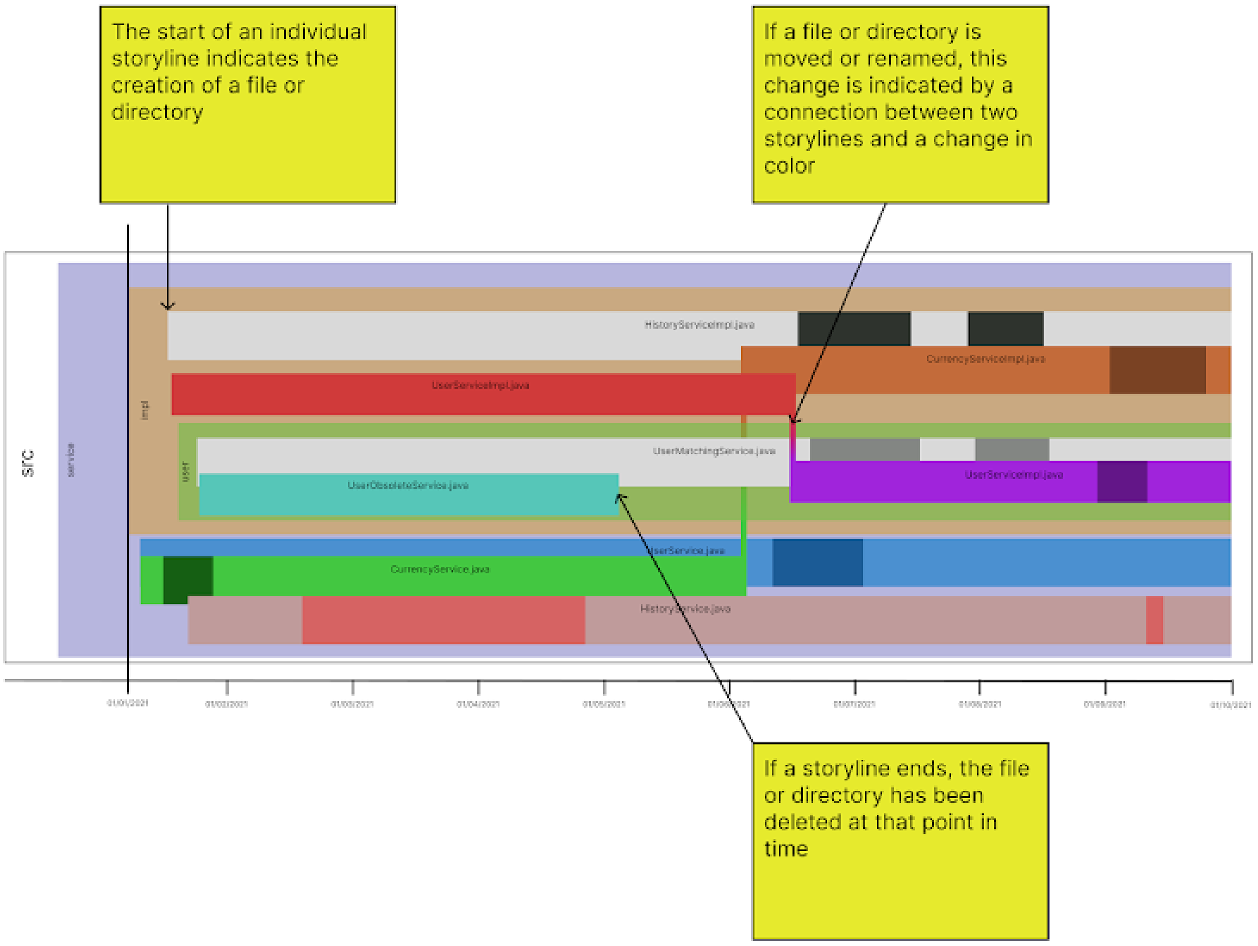
- ⬭ 0-3
- ⬭ 5-7
- ⬭ 7-10
- ⬭ 10-15
- ⬭ 15+

Visualizing Operations

On a more abstract level, files and directories overlap each other in order to provide the user wit a high-level overview of individual modules.


Files and directories in individual modules overlap in this abstract view to provide a general overview of the project
src
endpoint
mapper
persistence
service
test
unit
01/01/2021
01/02/2021
01/03/2021
01/04/2021
01/05/2021
01/06/2021
01/07/2021
01/08/2021
01/09/2021
01/10/2021
The timeline shows over which timeframe the project is visualized

As the user zooms in, more details about a specific module are revealed.



3. How clearly are the individual operations (create, move, delete) visualized? *

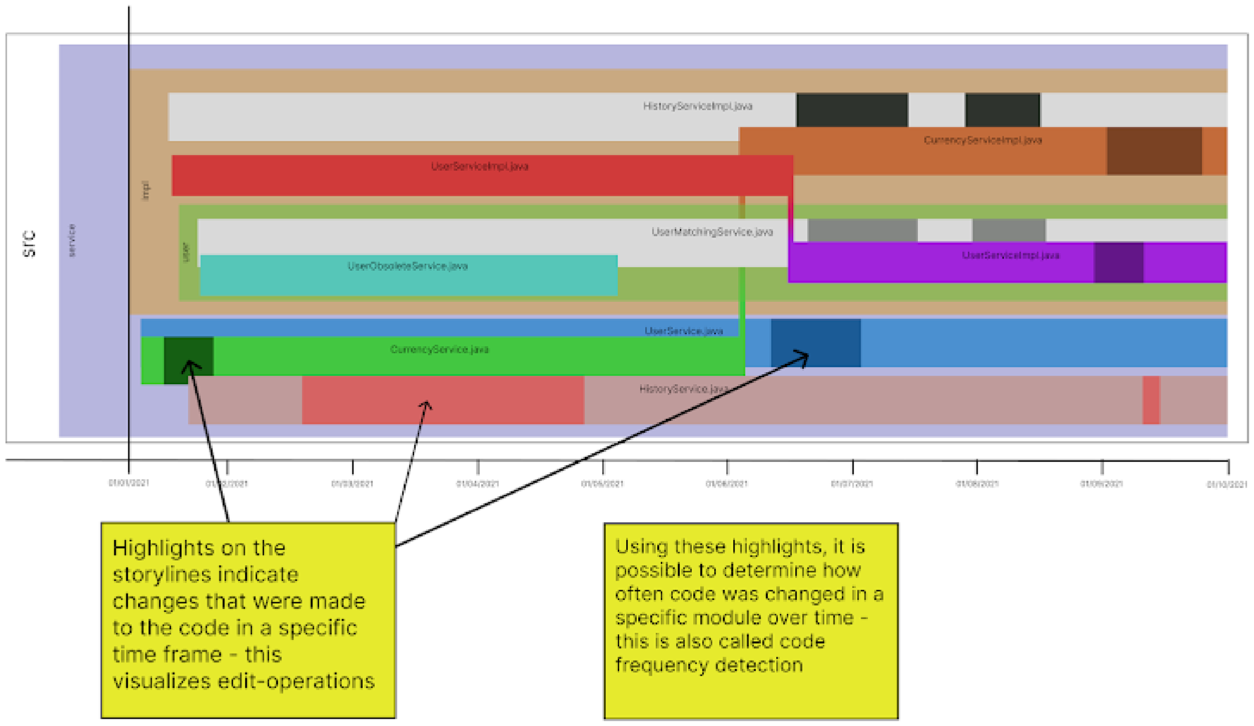
Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very clear

4. What could be improved when determining which operations were performed in the context of education?

Code Frequency Analysis

Using the visualization prototype, it is possible to analyze how often code was changed over tim in a specific module.



Highlights on the storylines indicate changes that were made to the code in a specific time frame - this visualizes edit-operations

Using these highlights, it is possible to determine how often code was changed in a specific module over time - this is also called code frequency detection

5. How useful is the visualization to determine at which points in time the code in a specific module changed frequently in the context of education?

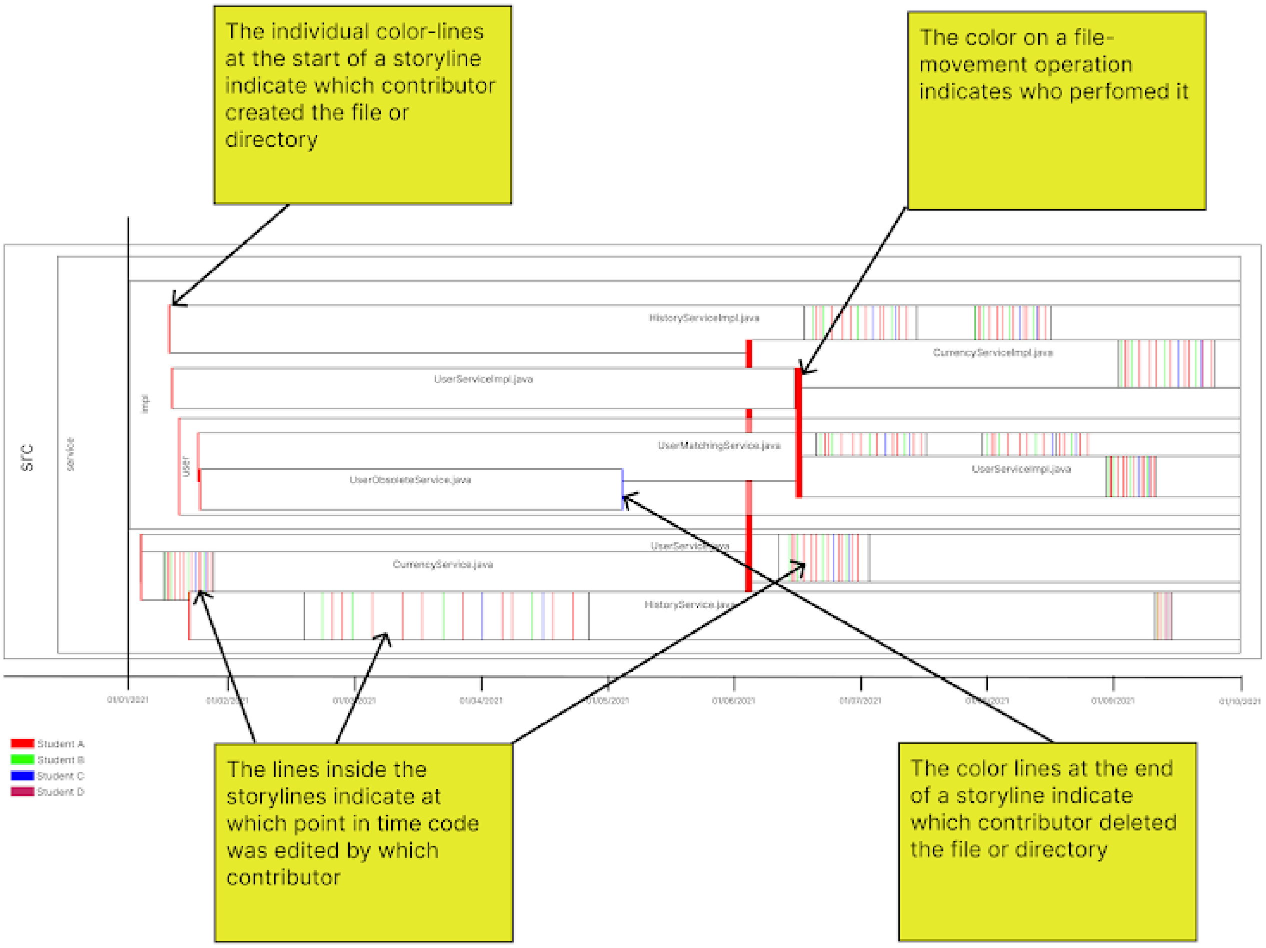
Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very useful

6. What could be improved with regards to code frequency visualization in the context of education?

Contribution Analysis

This view displays the contributions of each individual student in the given time-frame when looking at a student project.


The individual color-lines at the start of a storyline indicate which contributor created the file or directory
The color on a file-movement operation indicates who perfomed it
src
service
impl
user
HistoryServiceImpl.java
CurrencyServiceImpl.java
UserServiceImpl.java
UserMatchingService.java
UserServiceImpl.java
UserObsoleteService.java
UserService.java
CurrencyService.java
HistoryService.java
01/01/2021
01/02/2021
01/03/2021
01/04/2021
01/05/2021
01/06/2021
01/07/2021
01/08/2021
01/09/2021
01/10/2021
🟥 Student A
🟩 Student B
🟦 Student C
🟥 Student D
The lines inside the storylines indicate at which point in time code was edited by which contributor
The color lines at the end of a storyline indicate which contributor deleted the file or directory

7. How useful is the visualization to determine the contribution of individual students when evaluating their project?

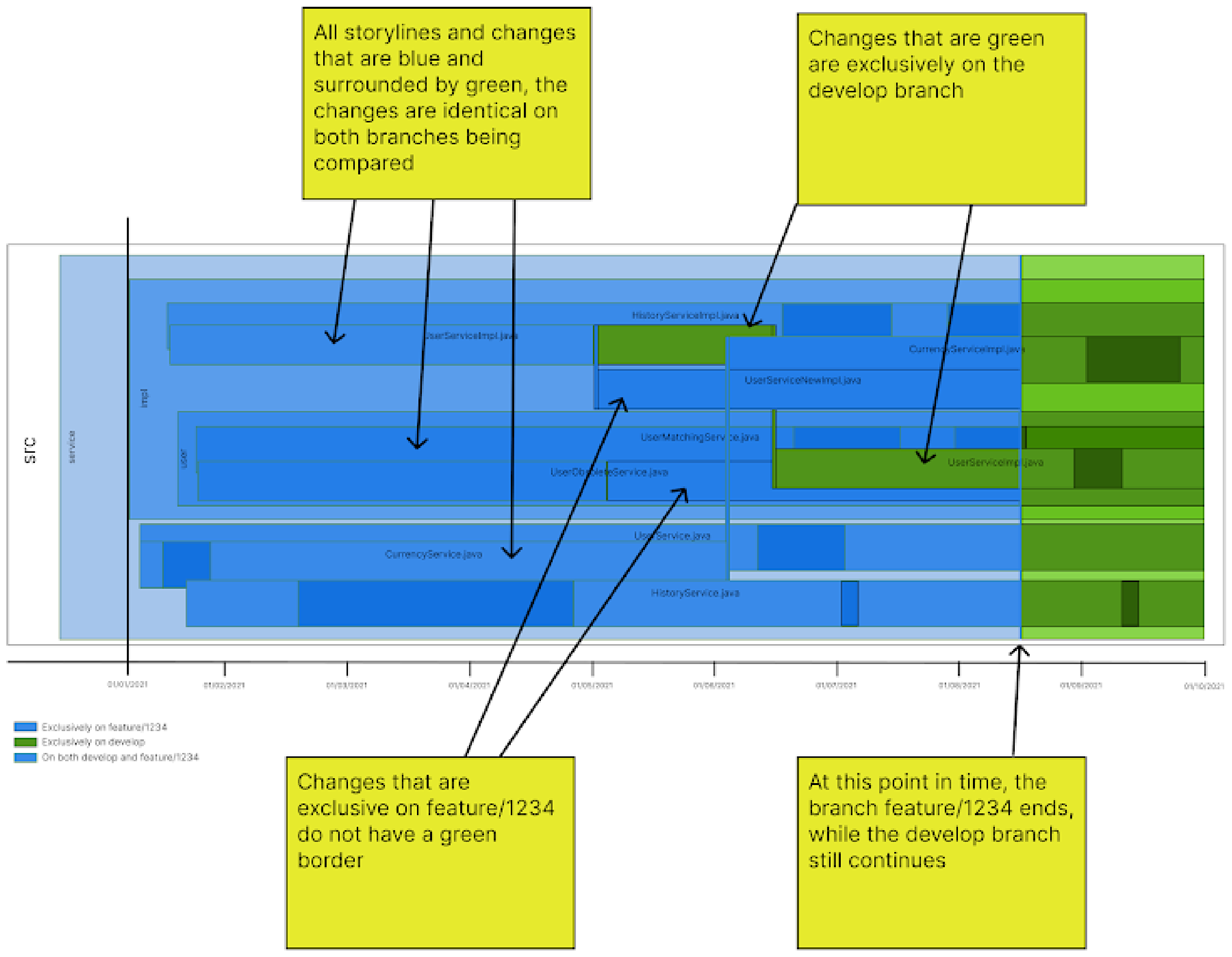
Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very useful

8. What could be improved when determining the contribution of individual students? *

Branch Comparison

This view compares two branches, in this case *develop* and *feature/1234* with each other. As the develop branch is more recent than *feature/1234*, it goes further in the timeline.The latest comm in *feature/1234* was made in the middle of August 2021.



All storylines and changes that are blue and surrounded by green, the changes are identical on both branches being compared
Changes that are green are exclusively on the develop branch
Changes that are exclusive on feature/1234 do not have a green border
At this point in time, the branch feature/1234 ends, while the develop branch still continues
src
service
impl
user
HistoryServiceImpl.java
UserServiceImpl.java
CurrencyServiceImpl.java
UserServiceNewImpl.java
UserMatchingService.java
UserObjectiveService.java
UserService.java
CurrencyService.java
HistoryService.java
🟦 Exclusively on feature/1234
🟩 Exclusively on develop
🟦 On both develop and feature/1234

9. How useful is the visualization regarding the comparison of branches in the context of education?

Mark only one oval.

	1	2	3	4	5	
Not	○	○	○	○	○	Very useful

10. Would this visualization be useful to analyze which changes were made during a large refactoring task performed by a student?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very useful

11. What could be improved when comparing two branches in the context of education? *

____

12. Do you see any additional use-cases for this kind of visualization for tutors or student assistants?

____

Overall Prototype

13. How useful is the visualization prototype in general to comprehend the evolution of the given software project in the context of education?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very useful

14. Which of the shown visualizations would you use in your grading process for student projects?

____

15. What could be improved in the overall visualization in the context of education?

16. Other feedback

This content is neither created nor endorsed by Google.

Google Forms

APPENDIX B

Prototype-First Survey

The following survey was used when participants started with the prototype. The evaluation design and results are presented in Chapter 6.

Visualizing Directory and File Evolutions for Software Repositories in Software Engineering Education

This survey evaluates the prototype visualization that was developed as part of the diploma thesis "Visualizing Directory and File Evolutions for Software Repositories in Software Engineering Education".

The prototype provides an overview over a software project by visualizing individual directories and files as storylines.
This way, it is possible to comprehend how, when and who changed which parts of the project architecture.
This can be utilized in an educational context by e.g. tutors and student assistants to quickly be able to evaluate a group project with regards to structure, past changes, code frequency, stale code detection and individual contributions.

By proceeding, you consent to participate in this study.
Your responses will remain confidential and will be used solely for research purposes. You may withdraw at any time.

* Indicates required question

https://master-thesis.danielkaufmann.at/

1. What is your role closest to? *

Mark only one oval.

- ◯ Software Developer
- ◯ Tutor
- ◯ University Assistant
- ◯ Project Manager
- ◯ Other: ____________________

2. How many years of professional experience do you have? *

Mark only one oval.

- ◯ 0-2
- ◯ 3-5
- ◯ 6-8
- ◯ 9-11
- ◯ 12-15
- ◯ 16+

3. How familiar are you with visualization tools for software repositories? *

Mark only one oval.

- ◯ 1: Not familiar
- ◯ 2: Slightly familiar
- ◯ 3: Moderately familiar
- ◯ 4: Very familiar
- ◯ 5: Extremely familiar

4. How familiar are you with the Git CLI? *

Mark only one oval.

- ◯ 1: Not familiar
- ◯ 2: Slightly familiar
- ◯ 3: Moderately familiar
- ◯ 4: Very familiar
- ◯ 5: Extremely familiar

5. How familiar are you with Git in general? *

Mark only one oval.

- 1: Not familiar
- 2: Slightly familiar
- 3: Moderately familiar
- 4: Very familiar
- 5: Extremely familiar

6. How familiar are you with IntelliJ IDEA, especially its Git integration? *

Mark only one oval.

- 1: Not familiar
- 2: Slightly familiar
- 3: Moderately familiar
- 4: Very familiar
- 5: Extremely familiar

Post-Task Questions: Prototype

Please indicate your level of agreement with the following statements regarding the prototype:

7. I think that I would like to use this system frequently. *

Mark only one oval.

	1	2	3	4	5	
Stro						Strongly Agree

8. I found the system unnecessarily complex. *

Mark only one oval.

	1	2	3	4	5	
Stro	⬭	⬭	⬭	⬭	⬭	Strongly Agree

9. I thought the system was easy to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	⬭	⬭	⬭	⬭	⬭	Strongly Agree

10. I think that I would need the support of a technical person to be able to use this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	⬭	⬭	⬭	⬭	⬭	Strongly Agree

11. I found the various functions in this system were well integrated. *

Mark only one oval.

	1	2	3	4	5	
Stro	⬭	⬭	⬭	⬭	⬭	Strongly Agree

12. I thought there was too much inconsistency in this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

13. I would imagine that most people would learn to use this system very quickly. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

14. I found the system very cumbersome to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

15. I felt very confident using the system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

16. I needed to learn a lot of things before I could get going with this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

17. Overall, how difficult were the tasks to complete using the prototype? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Difficult

Additional Feedback

Post-Task Questions: IntelliJ IDEA

Please indicate your level of agreement with the following statements regarding IntelliJ IDEA:

18. I think that I would like to use this system frequently. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

19. I found the system unnecessarily complex. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

20. I thought the system was easy to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

21. I think that I would need the support of a technical person to be able to use this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

22. I found the various functions in this system were well integrated. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

23. I thought there was too much inconsistency in this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

24. I would imagine that most people would learn to use this system very quickly. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

25. I found the system very cumbersome to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

26. I felt very confident using the system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

27. I needed to learn a lot of things before I could get going with this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

28. Overall, how difficult were the tasks to complete using IntelliJ IDEA? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Difficult

Additional Feedback

29. In comparison with the prototype, what advantages or disadvantages do you see when using IntelliJ IDEA?

30. Separately from that comparison, are there any disadvantages you noticed about the prototype itself?

31. In comparison with IntelliJ IDEA, what advantages or disadvantages do you see when usi
the prototype?

32. Separately from that comparison, are there any disadvantages you noticed about IntelliJ IDEA itself?

APPENDIX C

IntelliJ-First Survey

The following survey was used when participants started with IntelliJ IDEA. The evaluation design and results are presented in Chapter 6.

Visualizing Directory and File Evolutions for Software Repositories in Software Engineering Education

This survey evaluates the prototype visualization that was developed as part of the diploma thesis "Visualizing Directory and File Evolutions for Software Repositories in Software Engineering Education".

The prototype provides an overview over a software project by visualizing individual directories and files as storylines.
This way, it is possible to comprehend how, when and who changed which parts of the project architecture.
This can be utilized in an educational context by e.g. tutors and student assistants to quickly be able to evaluate a group project with regards to structure, past changes, code frequency, stale code detection and individual contributions.

By proceeding, you consent to participate in this study.
Your responses will remain confidential and will be used solely for research purposes. You may withdraw at any time.

* Indicates required question

https://master-thesis.danielkaufmann.at/

1. What is your role closest to? *

Mark only one oval.

- ◯ Software Developer
- ◯ Tutor
- ◯ University Assistant
- ◯ Project Manager
- ◯ Other: ______________________

2. How many years of professional experience do you have? *

Mark only one oval.

◯ 0-2

◯ 3-5

◯ 6-8

◯ 9-11

◯ 12-15

◯ 16+

3. How familiar are you with visualization tools for software repositories? *

Mark only one oval.

◯ 1: Not familiar

◯ 2: Slightly familiar

◯ 3: Moderately familiar

◯ 4: Very familiar

◯ 5: Extremely familiar

4. How familiar are you with the Git CLI? *

Mark only one oval.

◯ 1: Not familiar

◯ 2: Slightly familiar

◯ 3: Moderately familiar

◯ 4: Very familiar

◯ 5: Extremely familiar

5. How familiar are you with Git in general? *

Mark only one oval.

- ◯ 1: Not familiar
- ◯ 2: Slightly familiar
- ◯ 3: Moderately familiar
- ◯ 4: Very familiar
- ◯ 5: Extremely familiar

6. How familiar are you with IntelliJ IDEA, especially its Git integration? *

Mark only one oval.

- ◯ 1: Not familiar
- ◯ 2: Slightly familiar
- ◯ 3: Moderately familiar
- ◯ 4: Very familiar
- ◯ 5: Extremely familiar

Post-Task Questions: IntelliJ IDEA

Please indicate your level of agreement with the following statements regarding IntelliJ IDEA:

7. I think that I would like to use this system frequently. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

8. I found the system unnecessarily complex. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

9. I thought the system was easy to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

10. I think that I would need the support of a technical person to be able to use this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

11. I found the various functions in this system were well integrated. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

12. I thought there was too much inconsistency in this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

13. I would imagine that most people would learn to use this system very quickly. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

14. I found the system very cumbersome to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

15. I felt very confident using the system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

16. I needed to learn a lot of things before I could get going with this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

17. Overall, how difficult were the tasks to complete using IntelliJ IDEA? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Difficult

Post-Task Questions: Prototype

Please indicate your level of agreement with the following statements regarding the prototype:

18. I think that I would like to use this system frequently. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

19. I found the system unnecessarily complex. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

20. I thought the system was easy to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

21. I think that I would need the support of a technical person to be able to use this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

22. I found the various functions in this system were well integrated. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

23. I thought there was too much inconsistency in this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

24. I would imagine that most people would learn to use this system very quickly. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

25. I found the system very cumbersome to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

26. I felt very confident using the system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

27. I needed to learn a lot of things before I could get going with this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

28. Overall, how difficult were the tasks to complete using IntelliJ IDEA? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Difficult

Additional Feedback

29. In comparison with the prototype, what advantages or disadvantages do you see when us IntelliJ IDEA?

30. Separately from that comparison, are there any disadvantages you noticed about IntelliJ IDEA itself?

31. In comparison with IntelliJ IDEA, what advantages or disadvantages do you see when using the prototype?

32. Separately from that comparison, are there any disadvantages you noticed about the prototype itself?