FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

An Interactive Visualization of Software Quality Trends and Information Flows in Source Code Repositories

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

Benjamin Kowatsch
Matrikelnummer 0828124

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuer: Thomas Grechenig
Mitwirkung: Johann Grabner

Wien, 4. Mai 2019

(Unterschrift Verfasser) (Unterschrift Betreuer)

An Interactive Visualization of Software Quality Trends and Information Flows in Source Code Repositories

Master's Thesis

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media and Human-Centered Computing

by

Benjamin Kowatsch
Registration Number 0828124

elaborated at the
Institute of Information Systems Engineering
Research Group for Industrial Software
to the Faculty of Informatics
at TU Wien

Advisor: Thomas Grechenig
Assistance: Johann Grabner

Vienna, May 4, 2019

Statement by Author

Benjamin Kowatsch
Ufergasse 27/17, 3500 Krems an der Donau

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.

I hereby declare that I am the sole author of this thesis, that I have completely indicated all sources and help used, and that all parts of this work – including tables, maps and figures – if taken from other works or from the internet, whether copied literally or by sense, have been labelled including a citation of the source.

(Place, Date) (Signature of Author)

Acknowledgements

I want to thank Bianca Zinner for her patience, advice and motivational words which carried me through my whole study time.

I also want to thank my family for their endless support during my study time.

Furthermore, I want to thank my son who always made me smile at the end of the day. You are the real hero!

And last but not least, I want to thank my advisor whose support was invaluable for the success of this thesis.

Kurzfassung

Diese Diplomarbeit stellt eine Software-Visualisierung vor, welche der Analyse von Software-Qualitätstrends und der Ursachenforschung für mögliche Änderungen eines Trends dient. Die Idee basiert auf spezifischen Informationsbedürfnissen von Software-Entwicklern. Diese Informationsbedürfnisse wurden durch eine Analyse aktueller Literatur erhoben.

Bereits existierende Software-Visualisierungen zeigten, dass es hilfreich ist, sehr granulare Software-Qualitätstrends mit Versionsunterschieden von Quelltext zu verknüpfen. Dies ermöglicht es, einfacher auf relevante Quelltextänderungen zuzugreifen als dies mit bisherigen Software-Visualisierungen möglich ist. Außerdem werden dadurch die zuvor genannten Informationsbedürfnisse erfüllt.

Eine auf Szenarien basierte Evaluierung durch Experten hat gezeigt, dass der in dieser Arbeit vorgeschlagene Prototyp einer Software-Visualisierung das Nachvollziehen von Quelltext- und Qualitätsänderungen erleichtert und einen Mehrwert gegenüber aktuellen Lösungen bietet. Des Weiteren wurden die Szenarien durch Experten bewertet, um die praktische Relevanz des entwickelten Prototyps herauszustreichen. Auf der System Usability-Skala wurde der Prototyp mit “Gut” bewertet.

Schlüsselwörter

Trendanalyse, Qualitätsmetriken, Software-Visualisierung, Software Engineering

Abstract

This master's thesis proposes a software visualization that aims at analyzing software quality metric trends and identifying possible causes of change. The idea is based on the specific information needs of software developers that are hard to satisfy. These information needs were found with an analysis of the state-of-the-art literature.

A review of existing software visualization tools revealed that it is beneficial to combine fine-grained quality metric trends with code difference views based on data from version control repositories. This combination allows for easier access to relevant code changes compared to existing software visualization solutions and satisfies some of the previously mentioned information needs.

A scenario-based expert evaluation revealed that the proposed software visualization prototype makes the comprehension of code- and code-quality changes easier and has added value compared to current solutions. Experts were also asked to rate selected use cases of the prototype to emphasize its practical usefulness. On the System Usability Scale, the prototype is rated "Good".

Keywords

Trend Analysis, Software Quality Metrics, Software Visualization, Software Engineering

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

ACM Association for Computing Machinery

API Application Programming Interface

CLI Command Line Interface

CSS Cascading Stylesheet

HTML Hypertext Markup Language

IEEE Institute of Electrical and Electronics Engineers

JSON JavaScript Object Notation

NoSQL Not Only Structured Query Language

NPM Node Package Manager

PHP PHP: Hypertext Preprocessor

SHA Secure Hash Algorithm

SUS System Usability Scale

SVG Scalable Vector Graphics

UK United Kingdom

UML Unified Modeling Language

URL Uniform Resource Locator

VCS Version Control System

1 Introduction

In today's software development process, Version Control Systems are ubiquitous. Software development teams synchronize their work via these systems. During development, every code change impacts the quality of the software. Different tools allow to inspect the current status of a codebase regarding its history or to determine specific quality metrics [1, 50, 54, 61, 65, 69]. However, many of these tools are limited in at least one of the following:

Trend Analysis Most software visualizations tools offer no possibility of analyzing trends to understand the direction of quality in a software project. These tools usually focus on the current status of the codebase of a project. Trends allow different viewpoints of data, as values can be set in relation to each other.

Quality Metric Many tools do not incorporate quality metrics in their visualizations, but instead, focus on code and repository history.

Aggregation Quality metrics cannot be aggregated to understand relations or implications between multiple metrics.

Interactivity Visualizations often miss interactive components, for example, filter, search or tooltips.

Fine-Grained Structures Current tools often visualize metrics on project-level solely. Visualizations that also employ metrics based on single files, modules or even lines of code are less common.

Depicting quality measurements over time allows forming trends in visualizations. Trends provide a better overview than snapshots, as they allow to compare two or more values and set them in relation to each other. For example, test coverage of the project can be at a low at a specific point in time. This information does not provide any detail about how and especially why test coverage has developed this way. A trend of test coverage that goes over multiple commits from high to low may indicate other problems of the project. It enables to interpret data further and yields essential information for stakeholders involved in a project. A curve of a quality-metric from low to high and back to low could mean that a new developer has joined the project team. A fixed value of a quality-metric at a specific point in time is not able to provide such a range of data interpretation.

1.1 Motivation

Literature research has revealed that software visualization is an established field with many existing solutions. However, software visualization is not yet commonly used by software developers daily, according to Merino et al [46]. Comparing information needs to actual solutions reveals gaps in different software visualization categories, which might explain the low usage of software visualization [7, 41, 43, 46]. Figure 1.1 exposes the most notable gaps in software visualization categories defined by LaToza and Myers [41]. This thesis tries to satisfy information needs in the following categories:

Rationale Enables stakeholders to interpret quality data with trends so they can understand why a project or file has developed in a specific way.

Intent and Implementation Enables stakeholders to find out the intention behind a specific set of code changes and how it is implemented.

Refactoring Provides meaningful quality trends for stakeholders on a specific set of files to judge if refactoring is worth the effort.

History While this visualization category is not of high need in Figure 1.1, it is still essential to provide source code history to be able to find and inspect changes quickly.

Ko et al. [39] analyzed software developers' day-to-day information needs. Results of their research yielded developer questions that also fall into the categories defined above. Table 1.1 is a selection of questions found by Ko et al. [39].

Question	Category
What code could have caused this behavior?	Rationale
Is the problem worth fixing?	Refactoring
What is the purpose of this code?	Intent and Implementation
What have my coworkers been doing?	History
What code caused this program state?	Rationale, History
Why was the code implemented this way?	Intent and Implementation

Table 1.1: Software developer questions mapped to software visualization categories [39].

When looking at the above-defined categories and questions, it is interesting to know which kind of data is providing answers to these information needs. It is clear that the historical source code from the Version Control System (VCS) is of major interest in nearly every software visualization category. Combining historical source code data with quality measurements on each revision of the project's source code could lead to further insights why the quality of a project developed the way it did and may even include insights that lead to predictions for future developments.

Therefore, a visualization is proposed that displays a software project's quality data in different views. The commit view represents the average quality value of files on a per-commit basis. The file view does not average the quality value of files like the commit view but instead shows separate quality values for each file. In the module view, specific files can be aggregated to modules to generate quality values. Furthermore, a difference view is provided that does not render a trend chart, but instead can be used to find out exactly what lines of code were changed from one file version to another. Combining the visualization of quality data with the difference view makes changes comprehensible. This enables developers to recognize cause, in form of code changes, and effect, in form of changing quality metrics, in their source code.

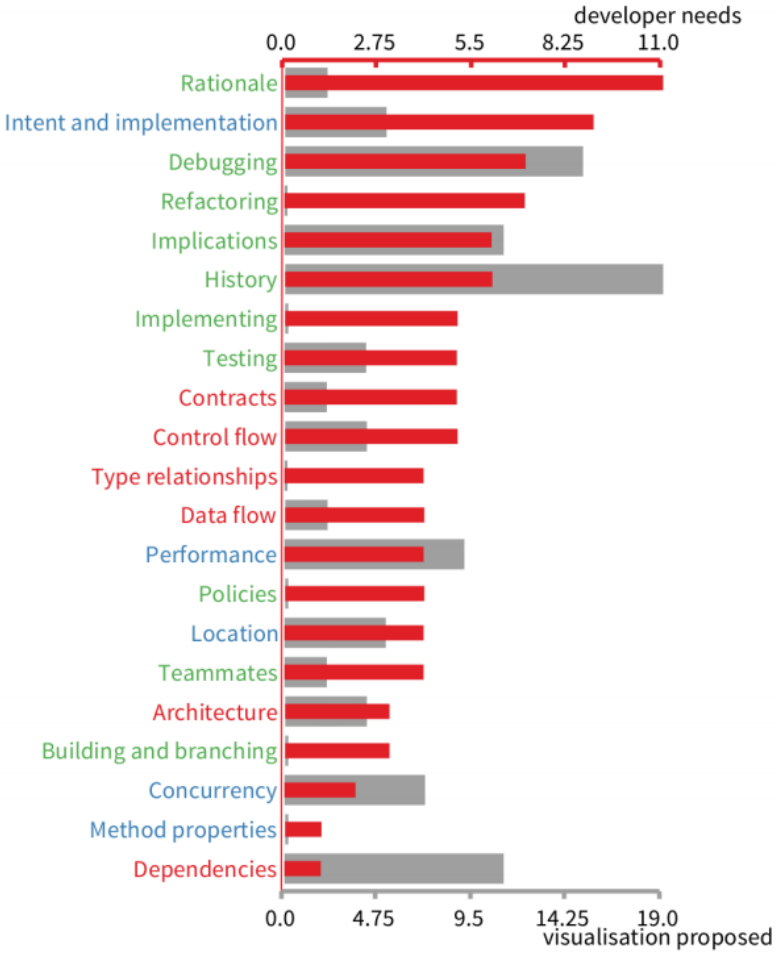


Figure 1.1: Comparing developer needs with proposed visualizations at IEEE VISSOFT conferences [46], [41].

1.2 Problem Description

Current visualizations of codebases hardly include quality metrics because they are usually focusing on other problem domains like for example code history [69], [61].

Existing tools, see Chapter 3, focusing on quality metrics and quality trends do not utilize visualizations as they are often static and not interactive.

For example, incorporating the possibility to select file versions directly within the visualization at specific points in time and compare them to each other to see why a quality value is changing seems like a trivial idea, however, none of the examined approaches is using this method of comparison.

Besides missing interactivity, quality trends in current visualizations are often only visualized on project-level solely and not for finer grained structures in the codebase, like single files or modules, see Figure 1.2 for an example implemented in SonarQube [63].

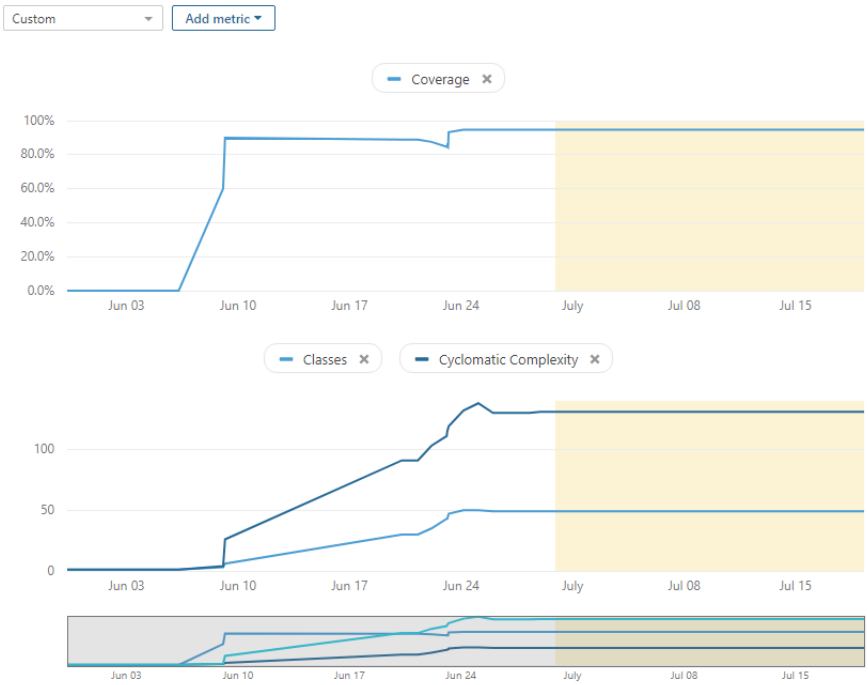


Figure 1.2: Visualization and comparison of three metric trends in SonarQube [63]. The metrics are visualized on project level.

Visualizations that do incorporate finer grained structures, like trends or lines of code, do not form trends over the computed quality metrics but instead often only show the current status of a project, see Figure 1.3 for an example implemented in PhpMetrics [44].

Figure 1.3: Visualization of Maintainability index and Cyclomatic complexity in PhpMetrics [44]. The size of a circle represents the Cyclomatic complexity and the color represents the maintainability of a file. This visualization displays a finer-grained structure of the project, its different files. However, it does not form trends over the history of these files.

Deriving from the stated problems, the hypothesis of this master thesis is stated as follows:

The visualization of fine-grained metric trends combined with code difference views satisfies practical information needs in software engineering.

1.3 Contributions

This thesis proposes a visualization method that focuses on quality trends of commits and files to support the decision-making process during a project's development cycle. Examples of software quality metrics are test coverage, nesting depth, number of functions as modularity, lines of code or Halstead metric [25]. Multiple metrics can be incorporated in the visualization as using software quality metrics in isolation is a common misapplication of quality metrics [20]. Stakeholders are able to analyze trends with the proposed visualization method by simply selecting the quality metric. This trend visualizes the development of the project regarding the selected quality metric. Selecting a specific commit in the visualization displays all associated files to this commit as additional trends. Files can be aggregated to modules. These trends visualize the finer-grained structures of the codebase. All these trends can be overlayed and enable further interpretation by comparing them. For example, if a file trend increases but at the same point in time the trend of the whole project decreases, the file is likely not the case of the decrease in the commit view. A detailed difference view for file versions complements the trend visualization.

The main contributions are:

Visual Trend Exploration The visualization enables stakeholders to recognize and analyze trends and understand the direction of quality in a software project.

Fine-Grained Quality Trends The visualization shows not only one aggregated view of the project, but also metrics for finer-grained structures within the project, for example, metrics based on single files or modules.

Multiple Trend Comparison Visualizing multiple trends simultaneously adds the possibility to compare trends and relate them to each other. Furthermore, trends from finer grained structures can be related to the project level and vice versa.

Improved Productivity The time it takes to analyze software quality data decreases for stakeholders. Also, the reasoning for changes in software revisions is more transparent by being able to see which influence a code change had on the software quality.

1.4 Methodology Outline

Following, a short outline of the employed methodology of this master's thesis is given. Starting with literature research, a requirement analysis as well as a technology review with a prototypical implementation is conducted. Consequently, a fully functional prototype is implemented and evaluated.

1.4.1 Literature Research

As a first step, literature research is conducted to get a better understanding of the information needs of stakeholders that need to be covered [7]. The comparison of different existing tools and visualizations ensures innovation of the visualization that is introduced by this thesis. The

VISSOFT conference[1] serves as the main starting point for the literature research. A systematic literature review is used to find and examine relevant literature [70].

1.4.2 Requirement Analysis

Following the literature research, requirements for the visualization are defined. As a starting point, the basic requirements are defined with the help of guidelines for analysis tools defined by Buse and Zimmermann [13]. To be able to judge the feasibility of the visualization, a prototypical implementation of these basic requirements is developed. After that, stakeholders that are impacted by the visualization are identified to incorporate their requirements for the visualization.

1.4.3 Technology Review

The next step is to find suitable technology for the proposed visualization. As already mentioned, a prototype is implemented, to find out which technologies help to fulfill the found requirements. As stated by Grady [27], it is important to assess the performance that the technology supports, otherwise there is a considerable risk that requirements cannot be achieved.

1.4.4 Implementation

The visualization is implemented as a client-server application. The prototype is iteratively refined, considering requirements and features for every revision. The features are based on the requirement analysis.

1.4.5 Evaluation

The visualization is evaluated by experts. Ten participants evaluate a prototypical implementation based on a list of tasks. Each task defines successful completion criteria that must be met before the task counts as correctly completed. Furthermore, participants rate each task based on their relevance in real-world environments. The evaluation is concluded with a rating of the prototype on the SUS [10].

1.5 Structure

Following, Chapter 2 addresses how literature research is conducted, how requirements are defined and how the technology stack is chosen. Chapter 3 covers the literature review and gives an overview of tools that are comparable to the proposed visualization. Chapter 4 defines the requirements for the prototypical implementation of RepoFlow. Chapter 5 explains how the visualization method is implemented and covers the design choices for implementation, architecture, and API. After discussing the implementation, Chapter 6 evaluates the visualization method and results of the evaluation are presented. The thesis is concluded with an outline of possible future work in Chapter 7.

[1] http://www.vissoft.info/

2 Methodology

The following chapter introduces the employed methods that were used to achieve the results stated in this master's thesis.

2.1 Research

As a first step to narrow down the subject of this thesis, a systematic literature review was conducted. According to Kitchenham [37], a systematic review is *a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest*. Budgen and Brereton [12] define the characteristics of a systematic literature review: a review protocol that lists the employed methods, a search strategy, documentation of the search strategy, inclusion and exclusion criteria as well as the information to be obtained from a primary study.

To conform to the characteristic of a review protocol, the techniques that were used are database search with the snowballing technique by Wohlin [70]. Figure 2.1 illustrates the approach. A start set of papers was chosen and then backward and forward snowballing was conducted. Backward snowballing uses references of a paper while forward snowballing uses papers that are citing the examined paper to find new papers. These papers were then taken into consideration if they fulfill specific criteria like publication date, citation count or keywords.

The search strategy of the literature review was as follows. A large part of the start set of papers for the snowballing technique were the Institute of Electrical and Electronics Engineers (IEEE) VISSOFT conferences from 2017 and 2018 as well as a review of selected papers introduced at previous IEEE VISSOFT conferences [46].

Additionally, a database search with Google Scholar, Association for Computing Machinery (ACM) digital library, IEEE digital library, and Springer was conducted. The keywords for the search were mostly derived from the title of this master's thesis as well as its hypothesis. They included "software quality," "software quality metrics," "software visualization," "software quality visualization," "software quality metric visualization," "trend visualization," "quality assurance software engineering," "information visualization," and "information needs software engineering."

The found papers were selected if they deal with the topic of software visualization. Naturally, there were not only scientific sources of software visualizations, as this is a field of interest not only relevant to the scientific community but also the software industry. Therefore, a wide variety of visualizations was found with a Google search.

An analysis of state-of-the-art tools and solutions combined the results from the literature research as well as the visualizations from non-scientific sources, highlighting their strengths and weaknesses.

Additional search terms were needed for Chapter 6, as it focuses on the evaluation of the visualization. As the evaluation of software and user research is a topic of its own, a separate database search with Google Scholar was conducted. The database search included the keywords "expert evaluation", "expert interview", "quantitative evaluation", "qualitative evaluation", "test plan", "usability engineering", "user research methods", "System Usability Scale", "usability test" and "usability questionnaire".

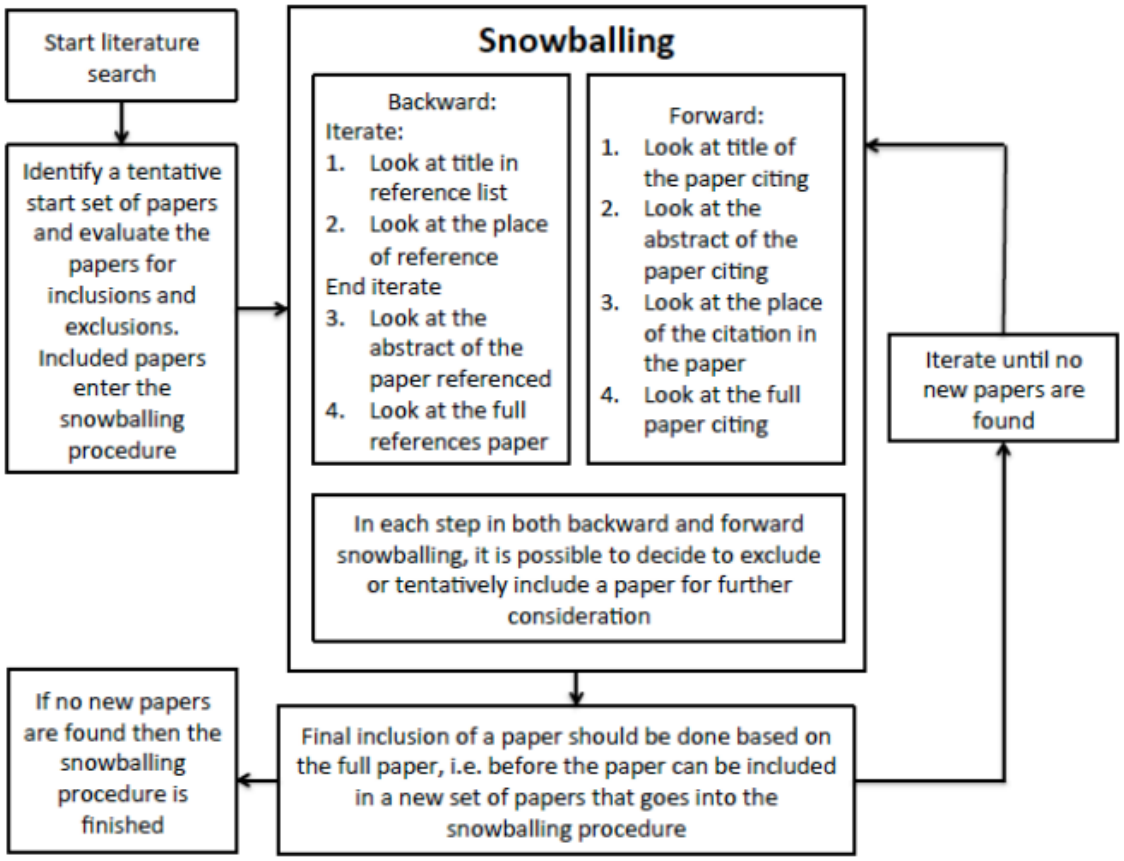


Figure 2.1: Snowballing technique for systematic literature review [70].

2.2 Requirement Analysis

Based on the analysis of state-of-the-art tools and solutions, basic requirements for the visualization were defined. After that, a prototypical visualization that implemented the basic requirements proved the feasibility.

Before the definition of the final requirements, stakeholders were identified. Stakeholders' needs and priorities have a profound impact on these requirements [24]. The final requirements were based on already conducted studies for software visualization and information needs [7, 13]. Finally, requirements were mapped to features of the visualization that implement these requirements.

2.3 Technology Review

As stated, a prototype of the implementation was built and served as the starting point for the validation of suitable technologies. To provide a cross-platform application, the visualization was implemented as a web application.

Based on the prototype, it became clear that a back-end will be needed for handling calls from the front-end, like retrieving all files for a specific commit, handle calls for creating custom quality metrics or setting the VCS configuration from a command line client.

The front-end will call the services provided by the back-end and builds the visualization based on the retrieved data. Therefore, it was important to factor in the data format that will be used

to communicate between the front-end and the back-end. The data format also influenced the selection of a framework for the visualization.

The database should also be able to easily handle queries and ideally serve the data format that the front-end uses for the visualization.

Today, JavaScript is used for both front- and back-end development to create "full-stack" applications that are entirely written in a single language [3]. The next step, therefore, was to decide which frameworks to use to build the back-end and the front-end.

As stated by Grady [27], it is important to assess the performance that the technology supports, otherwise there is a considerable risk that requirements cannot be achieved.

2.4 Implementation

The visualization was implemented as a client-server application. After the technology review, it was clear to use JavaScript's technology stack for the implementation to be as platform independent as possible.

The implementation of RepoFlow was conducted as an iterative and explorative approach. Starting with the idea to use Sankey diagrams for both the information flow of files between commits and quality metrics, it became clear that separation of these two aspects will be needed as to not overwhelm the user with information. For example, the flow lines between nodes overlayed each other if commits were too close to each other and as nodes grow with the number of lines of code, the actual quality value of the node on the y-axis became hardly recognizable. The separation resulted in the visualization of commits as balloons, files as trends and the information flow as a difference view between file versions.

Design decisions are critically reflected with the support of literature research. The decision to use trend or line charts is based on the time-oriented data type. Visual encodings like shapes and colors were also a factor in design decisions. The general structure of the visualization with different layers like commits and files was based on Shneidermans "Visual Information Seeking Mantra" [62], see section 5.6.1.

2.5 Evaluation

To evaluate RepoFlow's visualization methods, a scenario-based expert evaluation was conducted. As a first step, a test plan was created to prepare for the evaluation. The test plan defines goals, methods, research questions, participants and a task list with ten tasks. It served as preparation for the actual test. The tasks aimed at finding answers to the formulated research questions. Tasks consisted of a short description, a question, successful completion criteria as well as a benchmark.

To be able to unify the experience for each participant an introduction protocol was employed. The participant got a brief introduction as to what RepoFlow does. After that, each participant got introduced to the process of the evaluation.

Each participant conducted a demographic questionnaire. The demographic questionnaire focussed on the participant's experience with software engineering in general, software repositories as well as software quality metrics.

Then, each participant got a quick start guide in the form of a printed manual of RepoFlow and executed ten tasks from the task list A.1. The tasks were the same for each participant, but they were not executed in the same order to account for learning effects. The order of the tasks was defined by the Latin square design [59]. The tasks were chosen in a way so that all major features of RepoFlow were used by each participant.

To ensure the validity of the expert evaluation, it was important to select the right users as well as the right task [53]. All participants had a software engineering background and used repositories on a regular basis and had at least basic knowledge of software quality metrics.

Every participant was asked to think aloud during the execution of the tasks. This was crucial to get insight into the thought process of users while they were using the visualization as this could have identified what hinders them to complete a given task. Therefore, the focus of the expert evaluation was not primarily the time of execution but instead the successful completion criteria for each task.

With the participant's consent, audio and screen of the session were recorded for easier evaluation and reproducibility of the results. After finishing the tasks, each participant rated the previously executed tasks based on their relevance in a real-world environment.

Each participant concluded the expert evaluation with a follow-up interview where the participant was asked for general feedback and to fill out an SUS questionnaire. The interview aimed at finding out, which aspects of the RepoFlow visualization need improvement based on the participant's experience.

To verify the employed methods of the expert evaluation, two pre-tests were conducted before the actual evaluation. Participants of these pretests also had a software engineering background. They executed exactly the same process as regular evaluation participants. According to Porst [55], the execution of a pre-test is an inevitable requirement for a successful evaluation. This is because a pre-test uncovers problematic wordings or technical issues.

3 Related Work

Literature research has revealed that source code visualization is an emerging research field with many different solutions already existing. Providing innovation in this field needs grounded knowledge of already existing approaches. Due to the variety of software visualizations, there are also non-scientific approaches. To be able to better understand the presented approaches, the next section introduces relevant definitions. The following sections then introduce a selection of scientific and non-scientific approaches and explain the differences to the visualization introduced in this master's thesis.

3.1 Definitions

The following definitions are terms that are used in this master's thesis. These definitions do not only apply to this section but instead to the whole master's thesis.

Aggregation Mordal et al. [48] states that, *because most software quality metrics are defined at the level of individual software components, there is a need for aggregation methods to summarize the results at the system level*. As the employed visualization method in this thesis is based on files, the aggregation of quality metrics of commits is done by averaging the quality metric's value of all files that belong to the commit. One of the main disadvantages is that this approach does not convey the standard deviation [48]. However, as the visualization also allows to zoom in on the files that build the average for a specific commit, it is also possible to recognize potentially unwanted metric values of files in overall acceptable commits.

API This is the short form for application programming interface. APIs serve as an interface between developers and code that implements a certain functionality [51]. Generally speaking, every program is essentially an API as it usually consists of modular code and each module has an API [8].

Code Churn This quality metric measures the changes made to a component over a period of time [52].

Code Smell A definition of code smells is given by [66]. Code smells are a metaphor to describe patterns that are generally associated with bad design and bad programming practices. Originally, code smells are used to find the places in software that could benefit from refactoring.

Commit In a VCS, a commit stores the current changes to the repository [42].

Comment Lines This quality metric defines the number of comment lines within a file or project. A comment exists for the benefit of the programmer to explain certain pieces of code [14].

Cyclomatic Complexity This quality metric defines the number of independent paths within a program. An independent path is any path through the program that introduces atleast one new statement or condition [35].

Difference View A difference view is usually an editor to compare the content of two versions of a text file. It is used to synchronize code changes of different developers editing the same file. US Patent 9,430,229 [67] describes such a system.

File-level visualization In this master's thesis, a file-level visualization conveys information about data that is finer grained than project level visualization. It is used to visualize information about data from files or modules.

Issue Usually, software projects employ an issue tracker to distribute and keep track of the workloads within the project. Issues within an issue tracker can, for example, gather bug reports, feature requests or change requests [26].

Kiviat diagram/Radar Chart A Kiviat diagram, also called a radar chart, is used to display multivariate data. Kiviat diagrams can be described as radial transforms of stacked bar charts or line charts [31]. The data is usually visualized at intervals over 360 degrees.

Knowledge Monopoly Knowledge monopoly is an indicator of complex code that is only known to one specific developer. As complex code is hard to understand, a knowledge monopoly makes it hard to reorganize development teams [60].

Lines of Code This quality metric defines the number of lines of code within a file or project. This master's thesis mostly uses the definition of physical source lines of code. According to [68], *a physical source line of code is a line ending in a newline or end-of-file marker, and which contains at least one non-whitespace non-comment character*.

Parameters This quality metric defines the number of parameters in a method definition. Parameters are used to communicate informations to a method or function [47].

Project-level visualization or Commit-level visualization In this master's thesis, a project-level visualization or a commit-level visualization conveys summarized information about data from the whole project. It does not visualize information about data from more detailed structures within the project like folders, files or modules.

RadViz, Parallel Coordinates RadViz and Parallel Coordinates are high-dimensional visualizations to map n-dimensional data where $n \geq 2$ but preserving the visibility of all dimensions. Parallel coordinates can effectively display hundreds of dimensions and RadViz can display thousands of dimensions [17].

RepoFlow This is the name of the web application including all the components that this master's thesis introduces.

Repository A repository is the data structure in which a VCS stores information of a project [42].

Software Quality Metric According to the IEEE standard [5], a software quality metric is a *function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality*. Galin [23] states, that quality metrics should be included in three basic areas of software development: control of software development projects and software maintenance, support of decision taking and initiation of corrective actions.

Stakeholder This term is used according to the definition given by Glinz and Wieringa [24]. A stakeholder is a person or organization who influences a system's requirements or who are impacted by that system. In the context of this master's thesis, RepoFlow is the system that is influenced by specific persons or organizations.

Statements This quality metric defines the number of statements within a file or project. Source code usually consists of multiple statements. A statement changes the state of a program [9].

Static Code Analysis This is a software analysis approach that requires the actual source code, to evaluate it against multiple quality metrics. As opposed to Dynamic Code Analysis, the code does not need to be executed [18].

Technical Debt This is a metaphor designed for being communicated between engineering stakeholders and management stakeholders. It represents the future costs of early "quick and dirty" implementations in a software project [11, 15].

Trend For Trends, this master's thesis refers to an informal definition of trends by Robertson et al. [57]. A trend in data is an observed general tendency. The most common way to see a trend in data is to plot a variable's change over time on a line chart or bar chart. If there is a general increase or decrease over time, this is perceived as a trend up or down.

Usability According to Grudin [29], usability is given for a system that can easily be learned and handled.

Usefulness If a system provides usability as well as utility, usefulness is given [53].

Utility This is given if a system actually provides all needed features [29, 53].

VCS This acronym stands for version control system. A version control system maintains an organized set of all the versions of files that are made over time. Version control systems allow developers to go back to previous revisions of individual files, and to compare any two revisions to view the changes between them. In this way, version control keeps a historically accurate and retrievable log of a file's revisions [21]. Another explanation for version control systems is given by Spinellis [64]: "*with the version information that the VCS stores, you can access each file's history of changes, see the differences between versions of the same file, and see who changed which lines when.*"

Visualization Refers to the context of data visualization. According to [22], data visualization is the science of visual representation of "data", defined as information which has been abstracted in some schematic form, including attributes or variables for the units of information.

3.2 Scientific Approaches

This section lists all results obtained from the scientific community with a systematic literature review [70].

Chronos enables querying, exploration and discovery of historical change events to source code down to the line of code. Chronos does not aim at forming trends of quality metrics, however, its search and query functionality, as well as the visualization of the information flow between lines of code in files, is very powerful [61]. In Figure 3.1, a screenshot of Chronos' User Interface is shown where the difference between a file from one commit to the next commit is displayed.

Figure 3.1: Chronos - focusing on searching and querying of code and code history [61].

Chronicler explores visualization of the evolution of individual code elements, for example, functions, statements or variables. It provides another form of information flow that is not based on lines of code [69]. Figure 3.2 shows a screenshot from Chronicler. Different colors in the flow of information represent the addition of code elements, for example statements or variables. Chronicler does not focus on quality metrics within the project or the source code, however, it conveys the development of the structure of code over time in a powerful visualization.

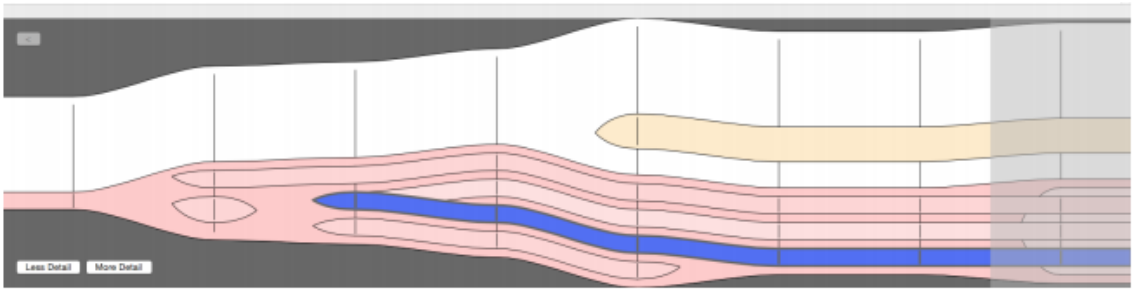


Figure 3.2: Chronicler - focussing on the code history of individual code elements [69].

The Code Time Machine tackles source code history based on files. All versions of source code files are placed one after another on the z-axis and can be brought to the front and checked out to the repository. Every file is opened in one tab in the application window if needed. Additionally, the code quality of each file version is represented as a trend chart on the z-axis. However, code quality is based on a per-file basis and not on a project level, as proposed with the commit view in this thesis [1]. Figure 3.3 shows a screenshot of The Code Time Machine with the most current version of a file in front of a file stack.

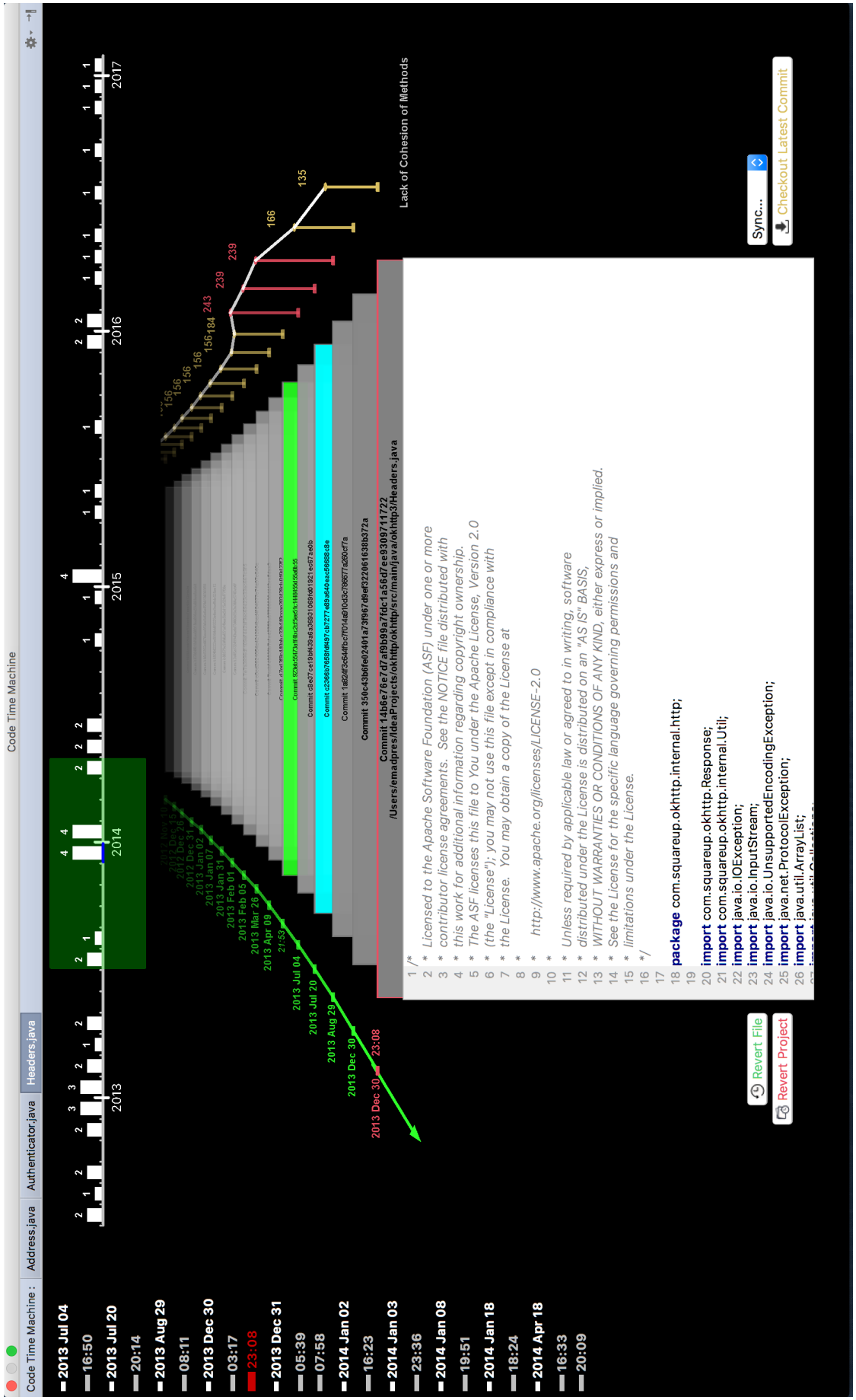


Figure 3.3: The Code Time Machine - Visualizing source code history with quality measurements on the z-axis as a trend based on files. The top area represents a bar chart with the number of commits. Zooming into this bar chart changes it to a timeline view of commits [1].

MetricView is a mixed 2-dimensional and 3-dimensional visualization method that combines a UML diagram visualization with metric visualizations. The main goal of this visualization method is to combine structural and metric information [65]. MetricView is based on the current status of the codebase. It is not possible to access historical quality data with MetricView.

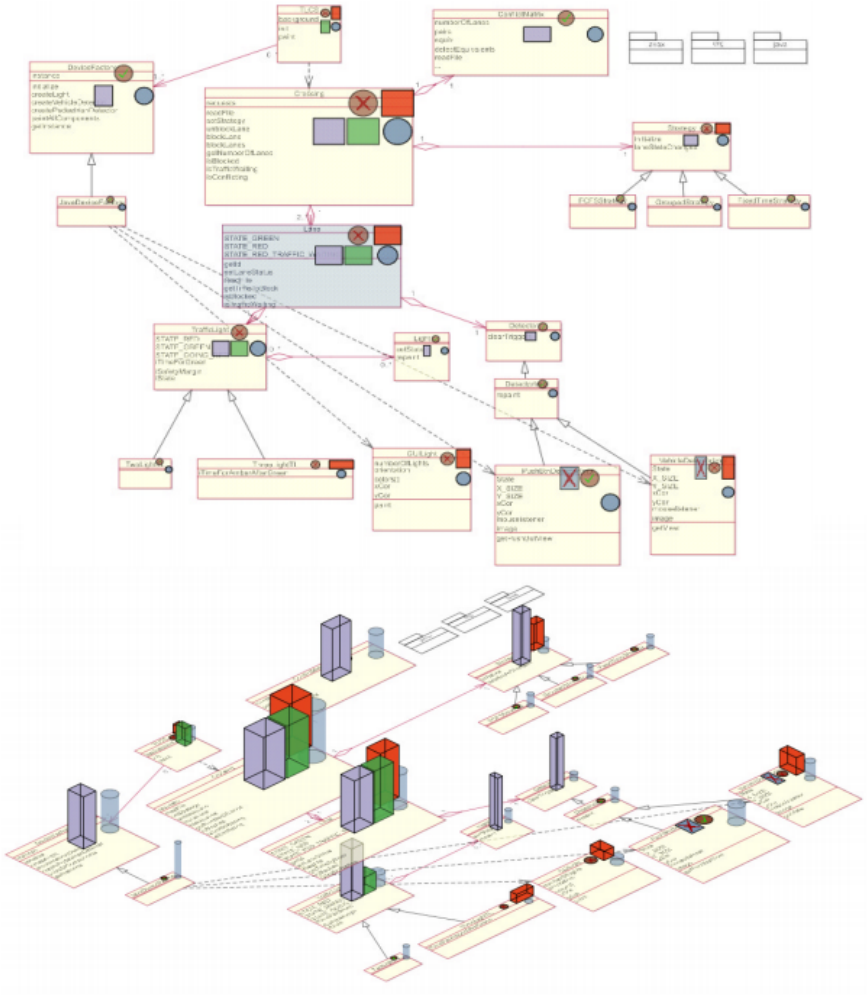


Figure 3.4: MetricView - 2-dimensional and 3-dimensional layouts. The structure is represented as a UML diagram and the metrics are represented by icons [65].

Exploring the Evolution of Software Quality with Animated Visualization is a visualization method consisting of 3-dimensional boxes laid out over a 2-dimensional plain. The boxes represent structural and version-control metrics. Additionally, Unified Modeling Language (UML) relationships are integrated into the visualization, similar to MetricView [65]. Data is retrieved from version control repositories, and the visualization can be navigated by versions. The presented visualization method aims at comparing metrics from one version to the other [40]. Figure 3.5 shows an example of the visualization where two classes are highlighted in different versions of the software.

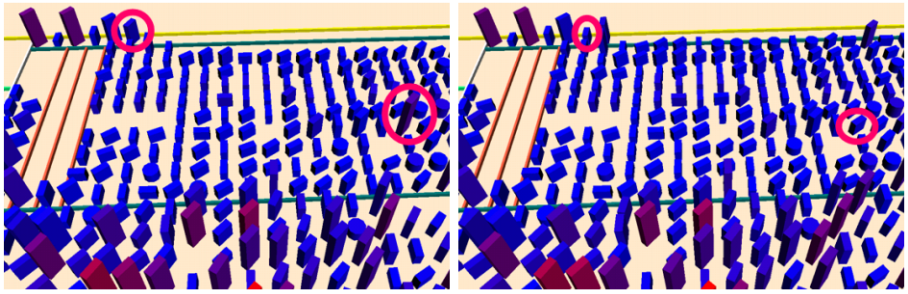


Figure 3.5: Exploring the Evolution of Software Quality with Animated Visualization - two classes highlighted in different versions of the software [40].

RelVis makes use of Kiviat diagrams to visualize quality metrics. Multiple Kiviat diagrams are linked with edges to indicate coupling dependencies between modules [54]. RelVis does not incorporate trends in visualizing quality metrics.

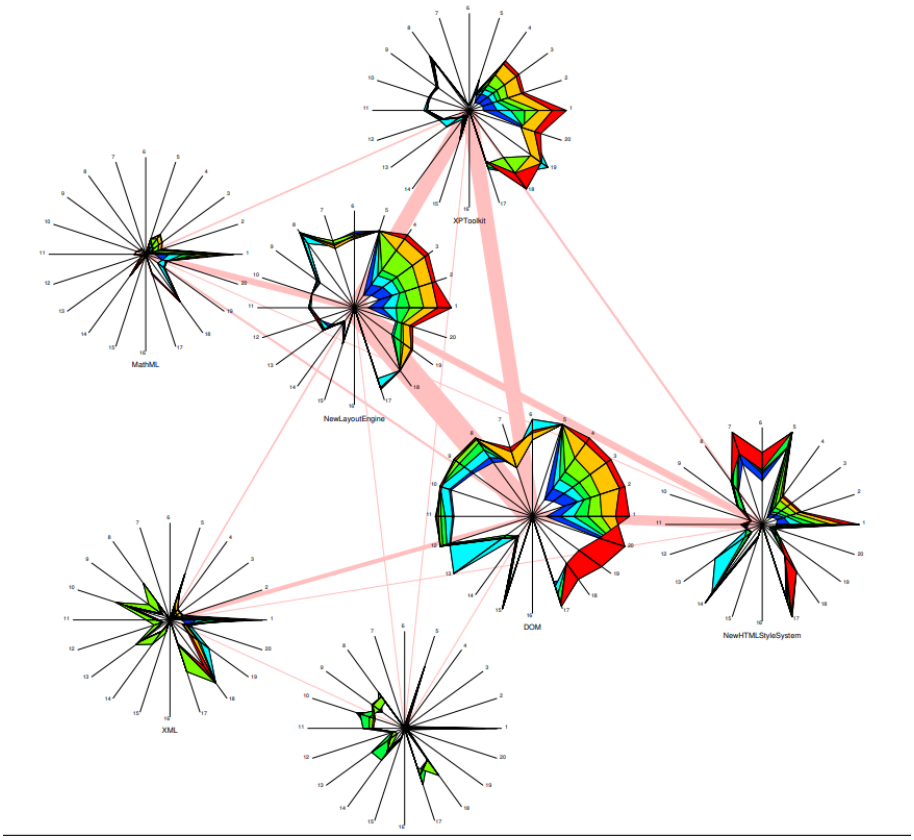


Figure 3.6: RelVis - Kiviat graph of 7 Mozilla modules implementing the functionality for handling the content and layout of websites. Each diagram presents 20 different source code and evolution metrics of software modules of 7 subsequent releases. Edges indicate coupling dependencies between the modules [54].

Stench Blossom also provides a view-based visualization of code quality but on the current version of a code file. The ambient view displays a petal on the right side of a code file in Eclipse that indicates either a low or high code smell (see Figure 3.7). Hovering over the petal switches to active view (see Figure 3.8) and displays the name of the code smell. A click in the active view switches to the explanation view (see Figure 3.10) and displays a detailed summary of the smell analyzer [50]. Stench Blossom does not incorporate trends or historical quality data.

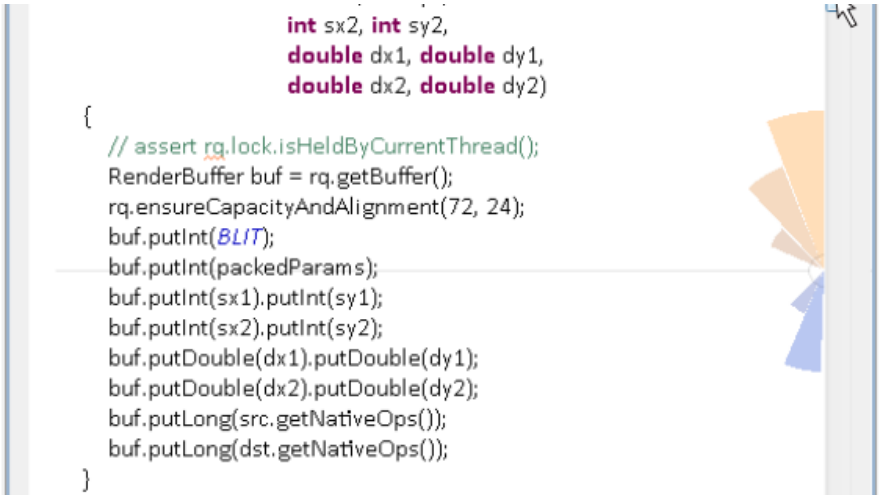


Figure 3.7: Stench Blossom - ambient view with petals on the right side indicating low or high code smells. This view is visible behind the program text whenever the programmer is using the code editor [50].

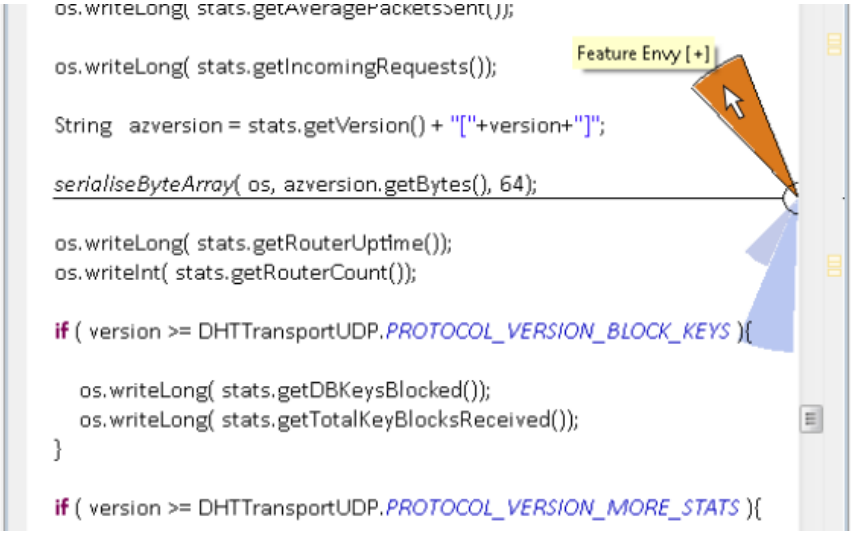


Figure 3.8: Stench Blossom - active view. Hovering over a petal in the ambient view activates the active view and displays the name of the offending smell [50].

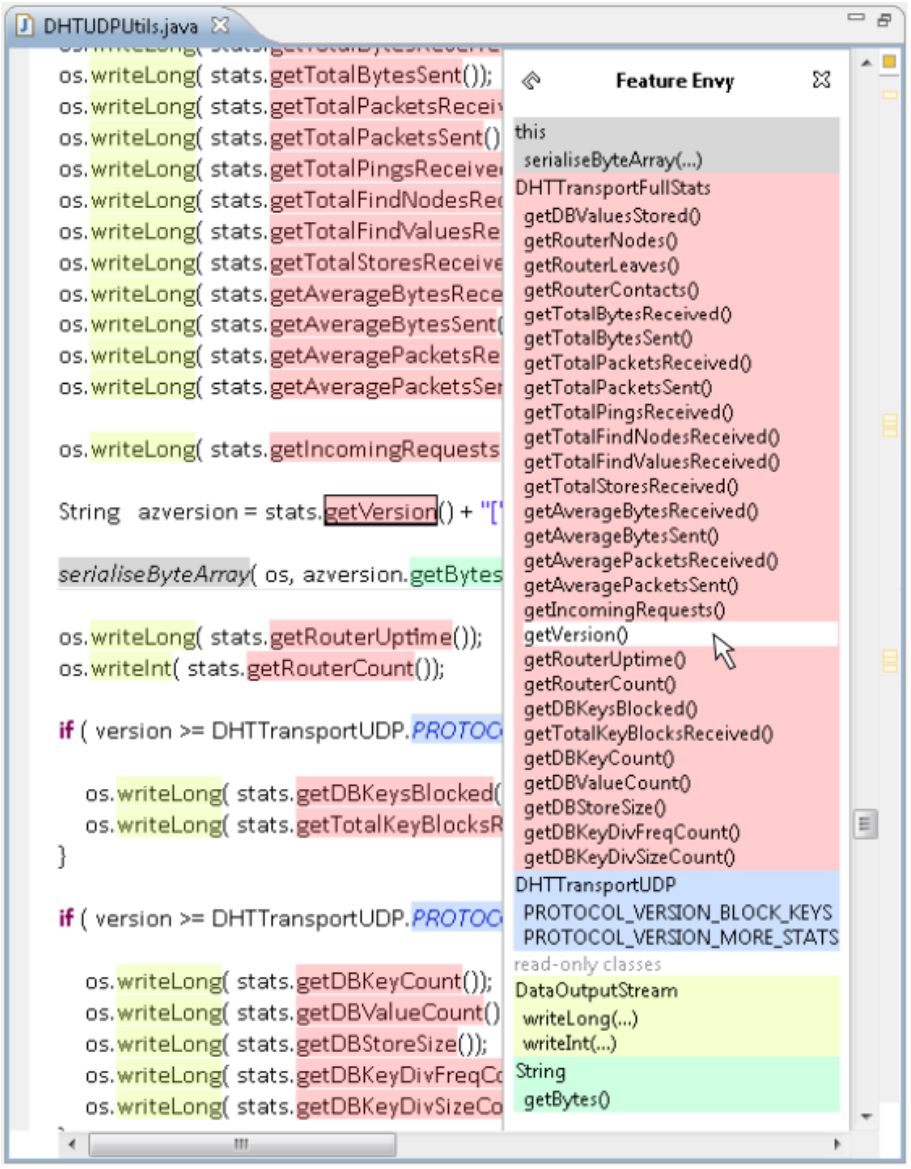


Figure 3.9: Stench Blossom - explanation view. This view explains the detected smell in more detail [50].

RepoVis is a tool providing a visual repository overview as well as search facilities [4]. The overview of RepoVis shows folders, files, and lines of code. The search functionality aims at finding sections by terms of interest within source code files, commit messages and metadata. The search matches are displayed visually in the overview. While RepoVis does not directly provide visualizations for code quality, searching terms of interest like “refactor” can point to commits, files or folders and can help to explore potentially problematic artifacts.

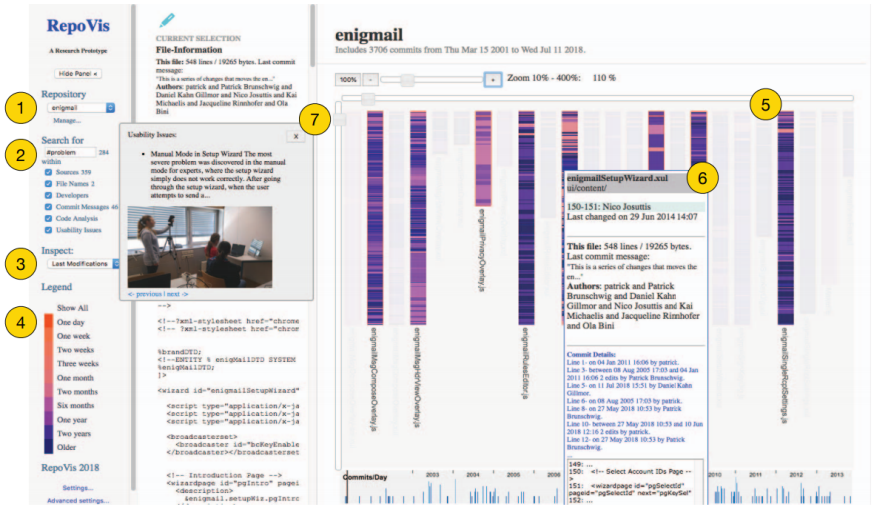


Figure 3.10: RepoVis - A typical software inspection with RepoVis. (1) A Git repository is cloned to the RepoVis back-end. (2) A full-text search is issued within all available search scopes. (3) A specific color mapping is chosen under Inspect. (4) The Legend allows filtering to particular facets. (5) Search matches are visualized in the overview. (6) Details are shown on demand for a particular line of code. (7) Usability issues or code analysis reports are shown for that line of code [4].

Linked Multivariate Visualizations Mumtaz et al. [49] are visualizing and analyzing multivariate software metrics with RadViz and parallel coordinates [17]. The goal is to automatically detect bad code smell patterns with the help of detecting outliers. Figure 3.11 gives an overview of the different parts of the visualization. Mumtaz et al. do not take historical data of the project into account, as opposed to RepoFlow's historical trend analysis.

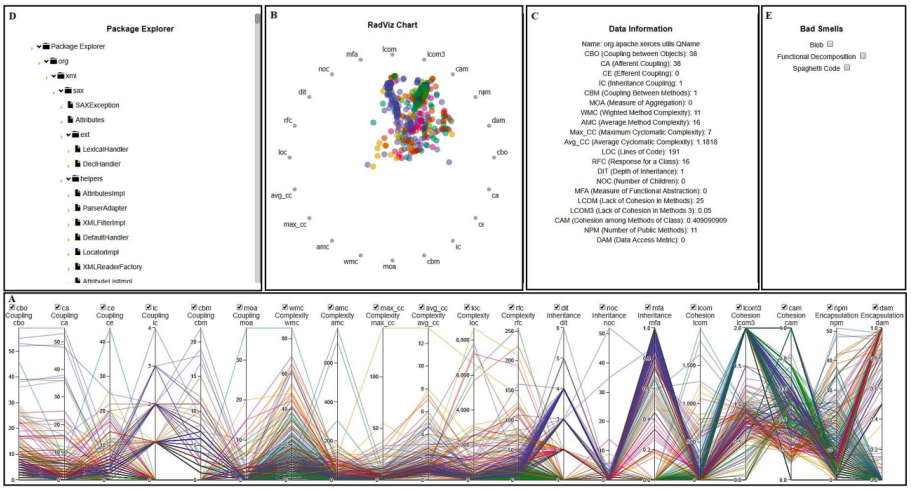


Figure 3.11: User interface of the solution of Mumtaz et al. (A) shows the parallel coordinates view. (B) is the RadViz view to explore noteworthy outlier patterns in detail with respect to a focused set of metrics. (C) shows software metrics details for a selected class. (D) is the package explorer for selecting packages and classes. (E) represents the options for automatic detection of basic bad smells [49].

Blended, Not Stirred: Multi-concern Visualization of Large Software Systems This visualization approach combines different data sources, like source code changes, bug tracking information, IDE interactions and stack traces. Furthermore, the visualization employs a set of metrics to each of the different data sources. As these metrics have different data sources, they are not the same as the ones employed by the visualization proposed in this thesis [16].

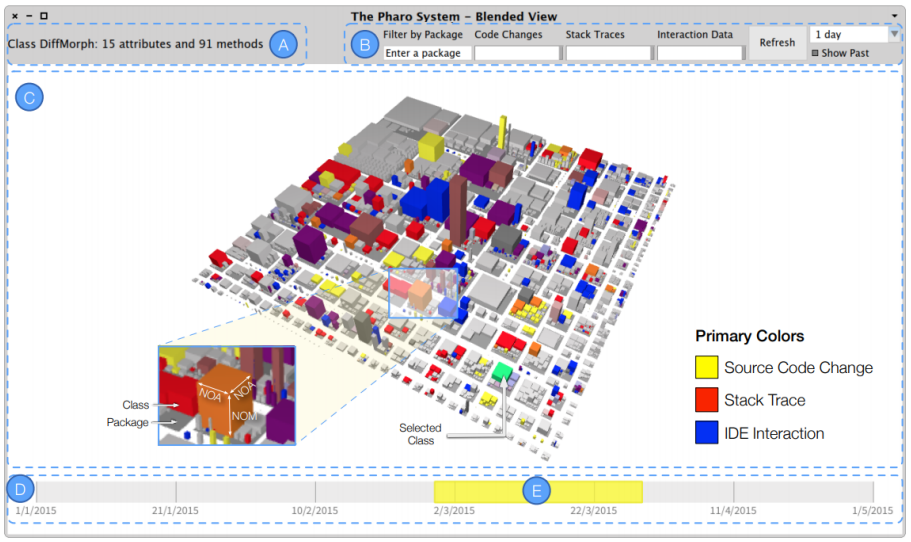


Figure 3.12: The Blended City: (A) represents a status bar to display additional information on the selected entity, (B) is a toolbar to customize the visualization, (C) is the view canvas, (D) is a timeline slider, and (E) represents a source code change on the timeline [16].

3.3 Other Approaches

As already mentioned, software visualization is an established field, and commercial tools also exist. These tools need to be examined to ensure innovation.

SonarQube aims at supporting stakeholders with continuous code quality. Its major features provide automated issue generation, security reports, quality metrics, and visualizations [63]. Visualizations of quality metrics are employed on the dashboard (see Figure 3.13) as well as in more detailed views (see Figures 3.14 and 3.15). Most of the visualizations show trends of the project either specific to one quality metric over time, or hot-spot graphs where quality metrics are related to each other. Compared to RepoFlow, SonarQube employs quality metrics only on the project level of trend visualizations. Also, SonarQube is not able to combine all metrics in its visualizations.

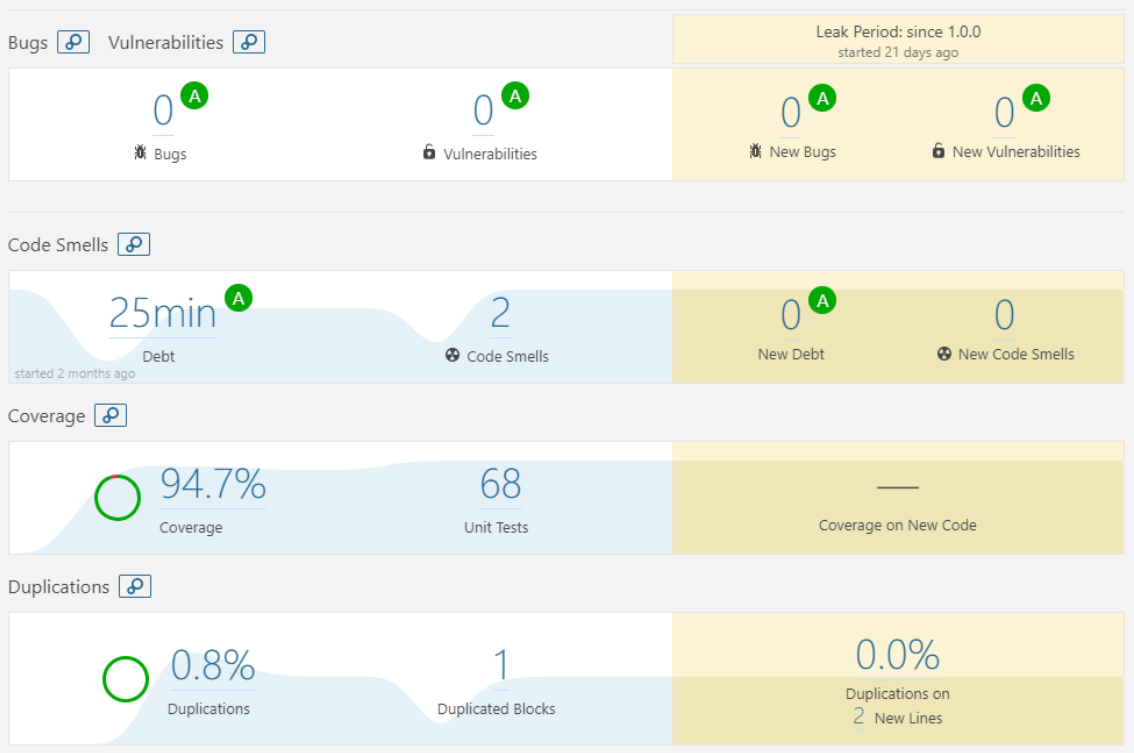


Figure 3.13: SonarQube - The dashboard shows quality metrics of the current status of the project and how these metrics trended since the start of the project [63].

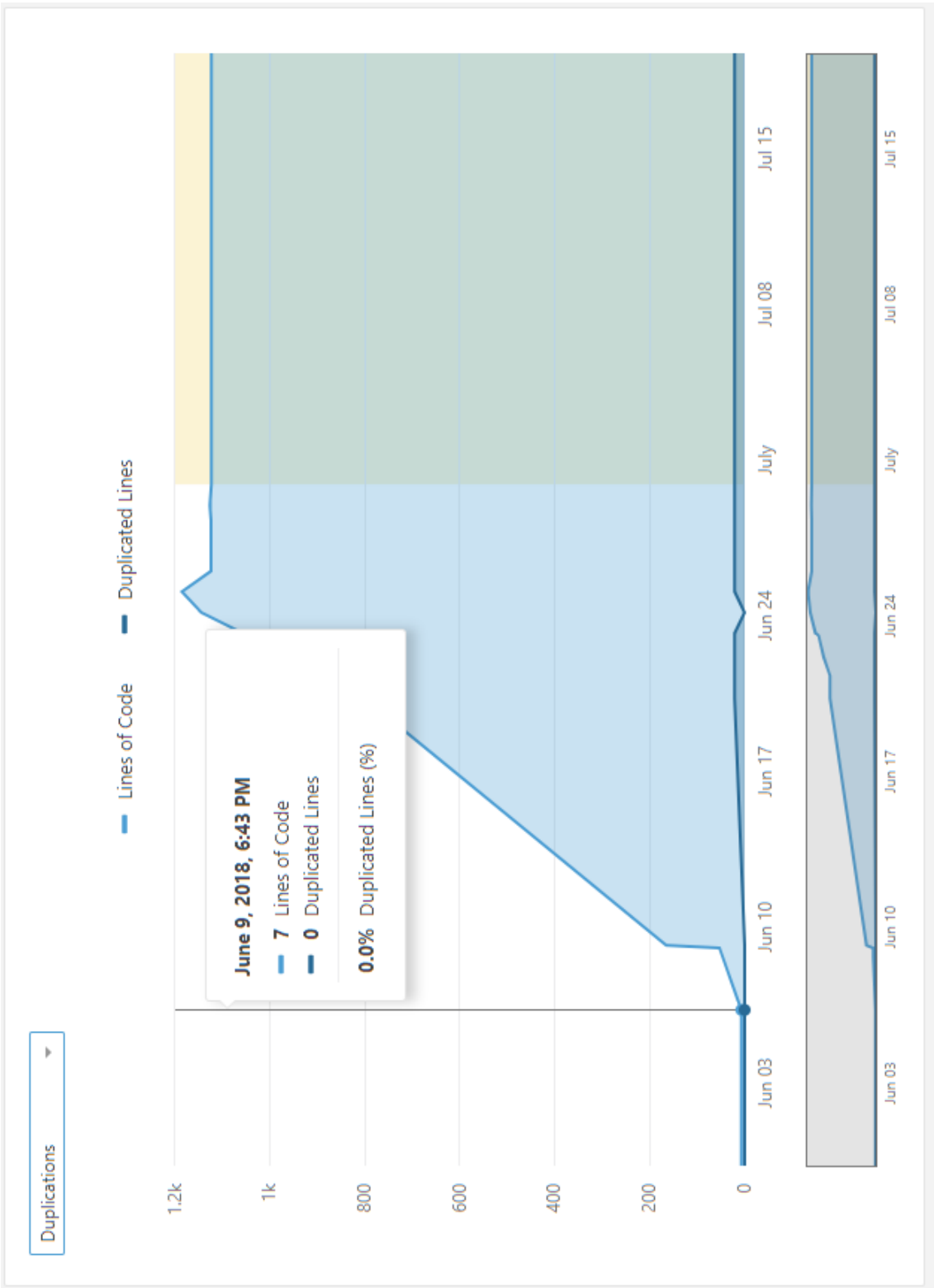


Figure 3.14: SonarQube - Trend of lines of code as well as duplicated lines in one graph to relate to each other. The y-axis represents the number of lines of code, while the x-axis represents time [63].

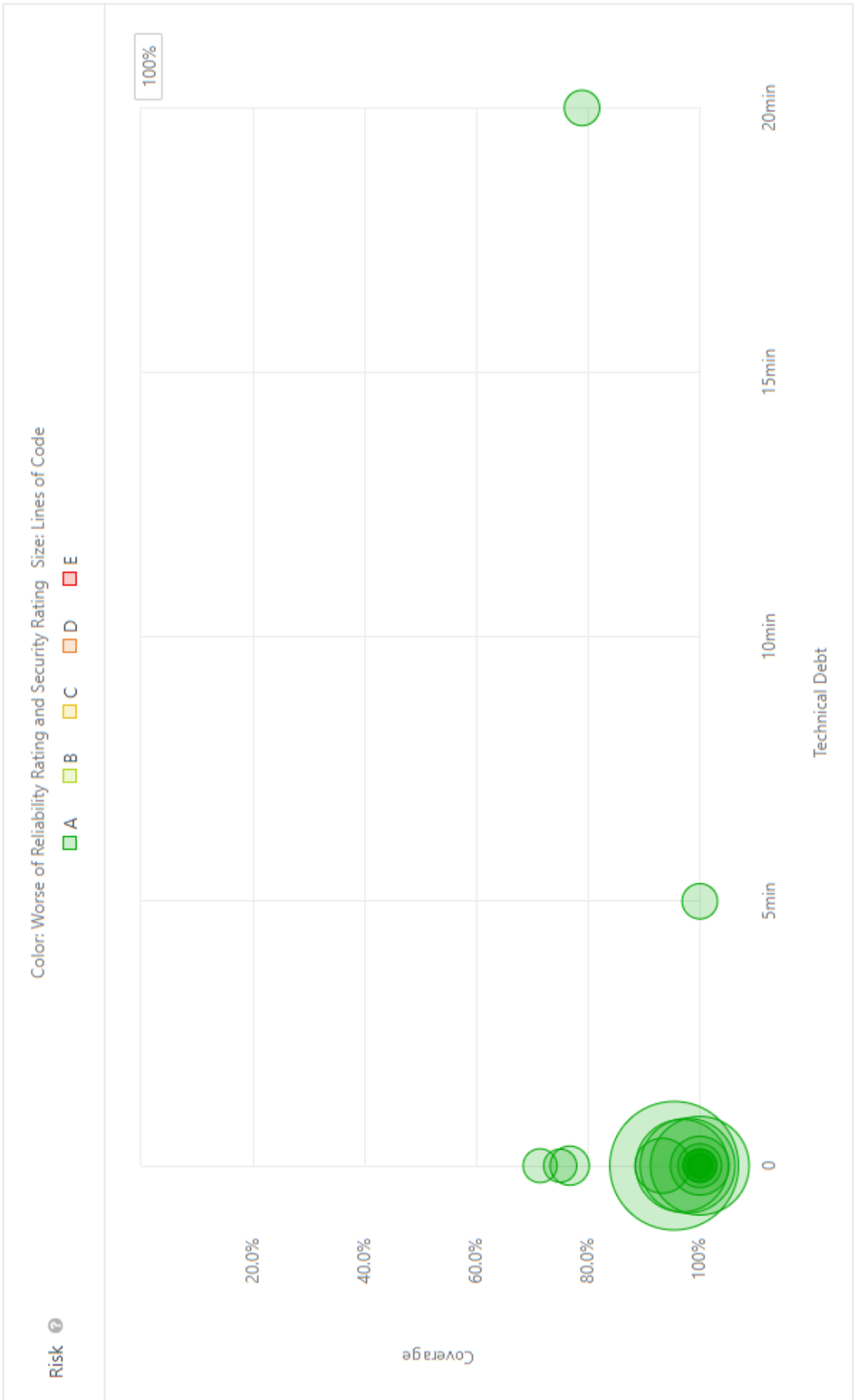


Figure 3.15: SonarQube - Technical debt of specific files as a hot-spot graph. The y-axis represents code coverage, while the x-axis represents the technical debt in minutes [63].

PhpMetrics is a static analysis tool for PHP: Hypertext Preprocessor (PHP) with a reporting feature that generates visualizations [44]. There are different visualizations like radial graphs for class coupling, clusters for maintainability and complexity, and hot-spot graphs that relate metrics to each other. The most important difference to RepoFlow is, that PHPMetrics is a static analysis tool. Its visualizations only represent the current state of the codebase and do not use data from a VCS.

Figure 3.16: PhpMetrics - Visualization of Maintainability index and Cyclomatic complexity generated with the PhpMetrics reporting tool [44]. The size of a circle represents the Cyclomatic complexity and the color represents the maintainability of a file.

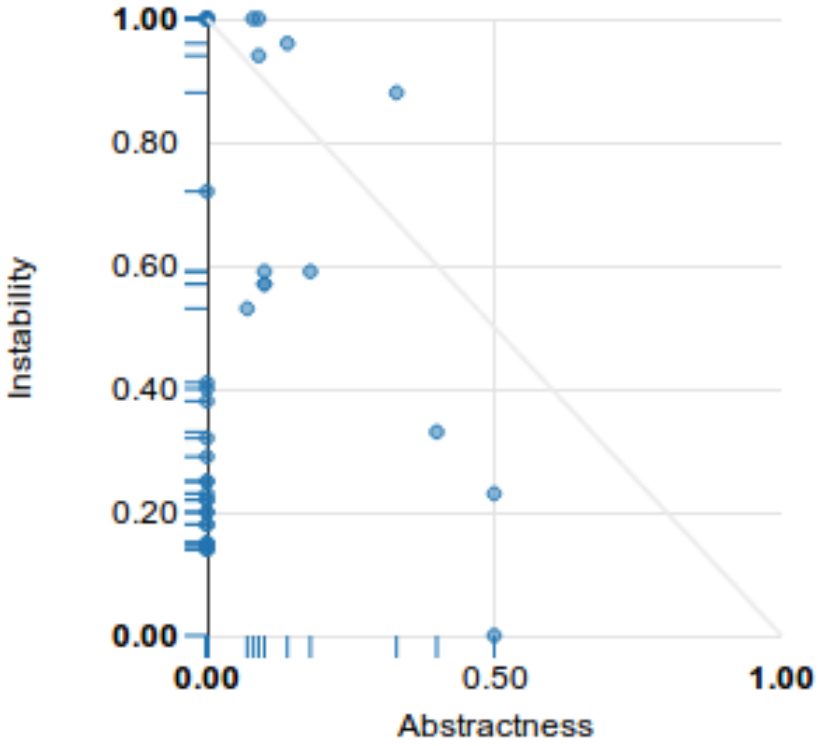


Figure 3.17: PhpMetrics - Abstraction Instability Chart generated with the PhpMetrics reporting tool [44]. Visualizes quality of software in terms of extensibility, reusability, and maintainability. The diagonal line is the Main sequence and packages are visualized as circles. The optimal values would be Abstraction = 1, Instability = 0 or Abstraction = 0, Instability = 1. Packages near this line have a good mix between those two metrics and are balanced. Other packages need attention [45].

Seerene tries to give stakeholders in management positions an analyzing tool to improve productivity and quality [33]. It creates metrics for different projects and enables to compare for example effort, complexity or knowledge monopoly over multiple projects. This enables stakeholders to relate projects to each other and decide which projects need attention in which areas based on quality metrics. For example, a project might have a high effort rate, but low technical debt. Another project has low effort rate but high technical debt and therefore needs improvement. With the visualization of trends at the project level and the combination of different metrics, Sereene can give hints about why a project developed in a specific way, similar to the visualization proposed in this master's thesis. Seerene also uses a 3-dimensional visualization of the folder and file structure where the height of blocks indicates problems in files. This master's thesis delineates mostly on the file level of the visualization, as it uses a 2-dimensional trend chart versus a 3-dimensional visualization of a selected point in time of the project. Also, the trend chart misses interactivity and multiple levels of abstraction. Seerene focusses on visualizations over multiple projects as opposed to RepoFlow that focusses on single projects.

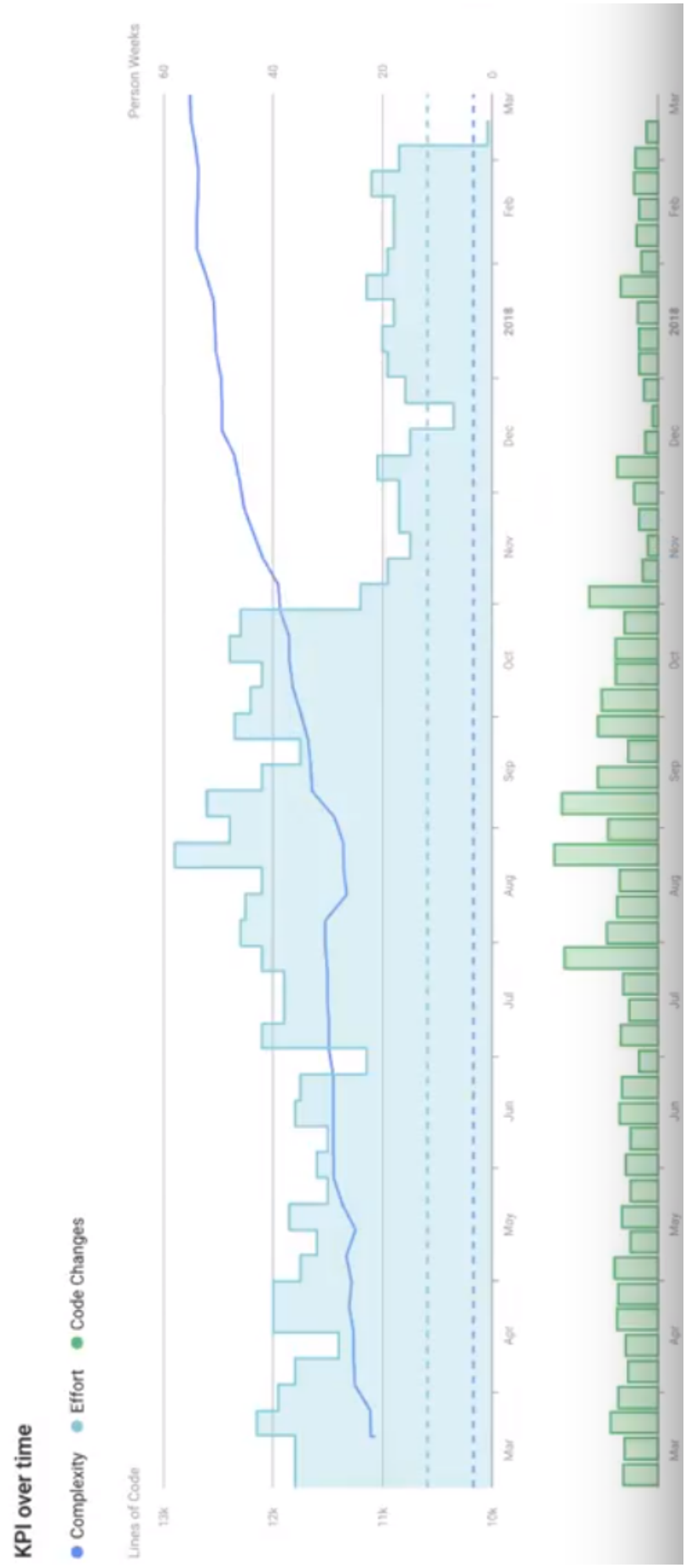


Figure 3.18: Seerene - Trending quality metrics: the blue trend line represents complexity, the light blue bars represent effort and the green bars at the top represent the number of code changes [33].

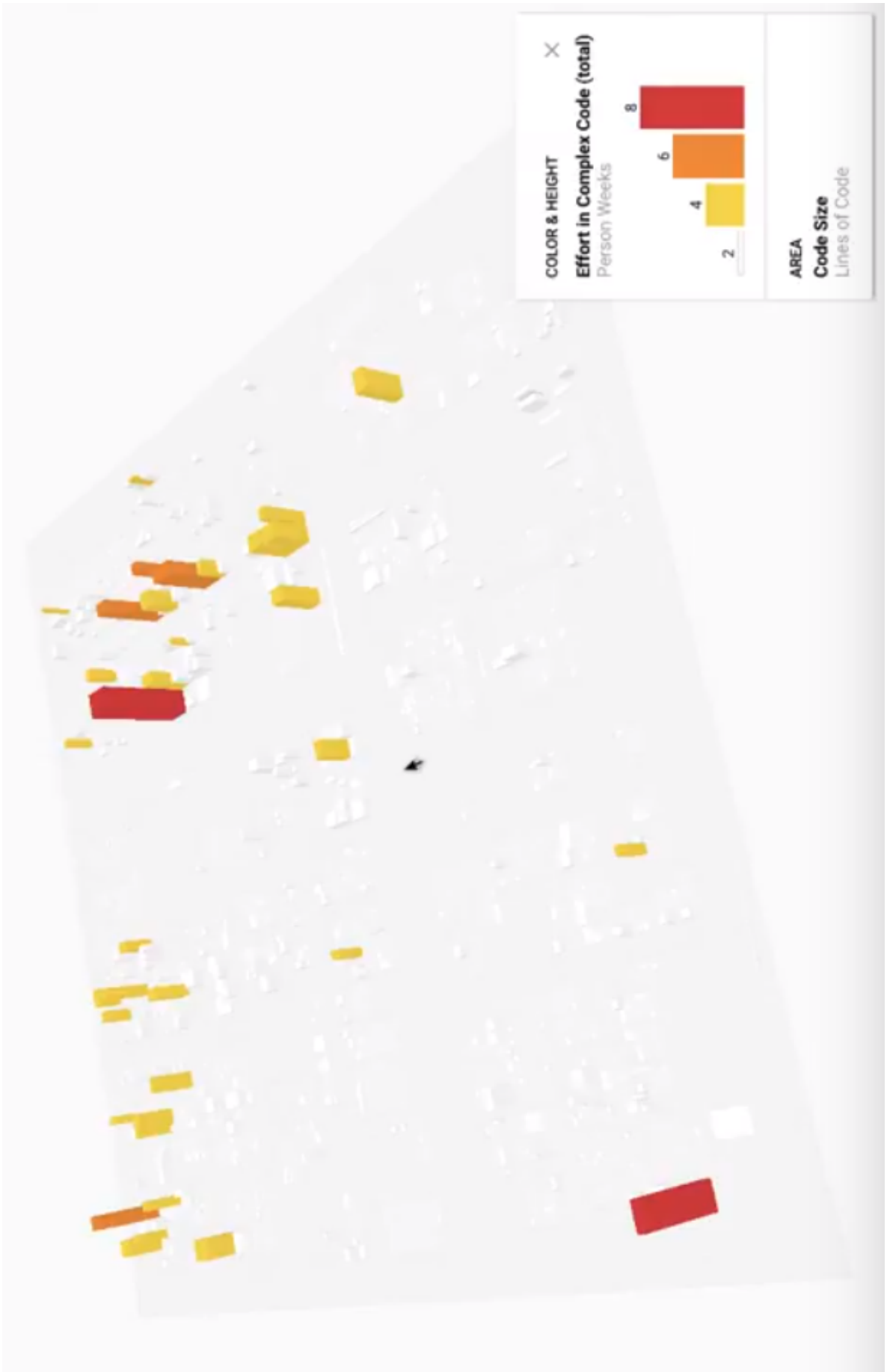


Figure 3.19: Seerene - Three-dimensional representation of the folder and file structure of a software project. High red blocks indicate problems [33].

Kiuwan also offers the possibility for cody analysis. Based on the analysis of defects, action plans are created in a continous manner [38]. One of the visulizations of Kiuwan includes a trend chart combined with a bar chart, that visualizes the defects in different action plans as well as removed defects, see Figure 3.20. Defects can be filtered by different categories and points in time. Kiuwan also provides a governance dashboard providing the most indicative metrics at a glance. Kiuwan offers the possibility to drill down through multiple levels of a project but only for the current status of a project, opposed to RepoFlow that forms a trend for quality metrics over multiple revisions.

Figure 3.20: Kiuwan - Visualization of defects in action plans as a bar chart. Removed defects are visualized as a trend chart [38].

4 Requirement Analysis

As mentioned earlier, software developers have a high need of software visualization categories like Rationale, Intent and Implementation, Refactoring and History. Also, problems of current visualizations of quality metrics were outlined in Chapter 1. This chapter introduces requirements that aim to satisfy these strongly needed categories combined with solutions to the mentioned problems. This leads to a visualization that answers questions that cannot or just partly be answered with existing tools and approaches (see Chapter 3).

4.1 Basic Requirements and Feasibility

For the definition of a basic set of requirements for the visualization, different sources are taken into consideration. First, the revealed gaps in section 1.1 and 1.2 serve as an indicator of problems that the visualization addresses. Second, guidelines for analysis tools defined by [13] are employed. These guidelines are as follows:

- Be easy to use for stakeholders that not necessarily have expertise in analysis.
- Be fast and produce concise or summary output. Stakeholders may have significant time constraints.
- Measure many artifacts using many indicators. Many are important and combining them can yield more complete insights.
- Be current and interactive. Stakeholders want to view the most current data available, at many levels of detail, not static reports.
- Focus on characterizing the past and present over predicting the future.
- Recognize that managers and developers have different needs and focus on information relevant to the target audience.

Based on these sources, the basic requirements for the visualization are as follows:

Trends, Quality and Time The specific quality value of a file or commit must be visible for a specific point in time. However, the nominal value of a quality metric at a point in time is often less important than how it is changing or "trending." Many decision scenarios describe intervening when negative trends are detected [13]. Trends enable assertions like "a commits quality value is trending up" or "a commits quality value is trending down." Additionally, trends make it possible to relate one quality value at two or more specific points in time. It is possible to say "this quality value has increased by 60 percent from 22.11.2017 to 01.12.2017 and therefore has risen dramatically." This leads to understanding and evaluating past decisions as well as making informed decisions about the future.

Granularity To provide visualizations of a large variety of data, the visualization must provide different levels of detail. This is achieved by providing project-level visualizations, file respectively module visualizations and a difference view for selected files. To provide a

time range that can be adjusted from granular to coarse, the visualization should allow to zoom in and out on the time axis.

Interactivity To be able to provide a wide range of granularity, the visualization must be highly interactive. It is not the goal to develop a static report, as the visualization should be able to provide a large range of data that can be selectively displayed. This also includes the ability to filter files or commits by specific quality metrics. To keep the visualization clear, there should be options to reduce the opacity of visualized elements or remove them entirely.

Combine Quality Metrics This is the most important requirement of the visualization. Quality metrics are especially expressive when combined. For example, code churn might have risen for a few commits. But why did that happen? Plotting the trend for *active developers* shows that three more developers committed during the last two weeks. These three developers were new employees and have a higher *code churn* rate during their teaching phase. Visualizing only one quality metric, code churn in this case, would not have led to that insight.

The feasibility of the visualization is verified with a simple prototypical visualization (see Figures 4.3, 4.2, 4.4) and the basic set of requirements. A server is set up with mocked data of a simple VCS, and the visualization fetches data from the server and displays it as a trend chart with quality and time values.

The visualization is based on data from version control systems and focuses on the quality components of the data. It will depict the flow of information over time on the x-axis and a scale maps the quality of the data on the y-axis. The idea of the flow of information is drawn from Sankey diagrams. Sankey diagrams are traditionally used to visualize the flow of energy or materials in various networks and processes [56]. Figure 4.1 shows the scenario of energy supplies and demands in the United Kingdom (UK) in the year 2050 as a Sankey diagram[1].

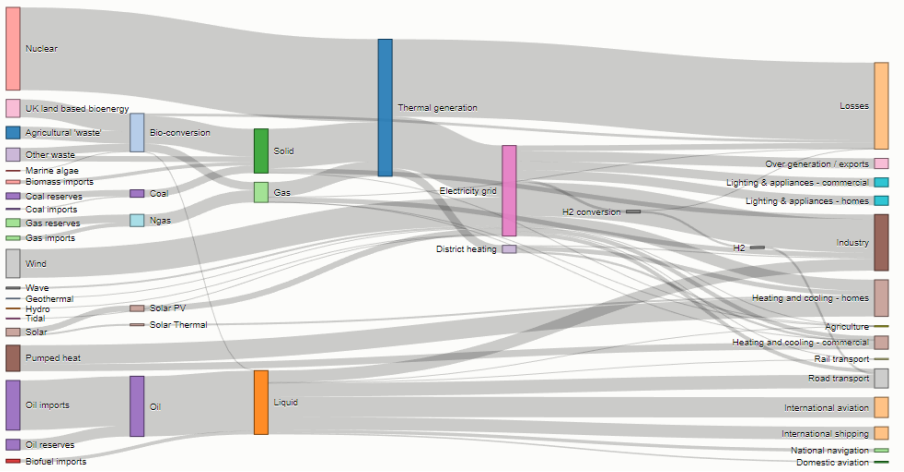


Figure 4.1: Example of a Sankey diagram[1].

The main drawback of Sankey diagrams is the lack of a proper visualization of the time dimension. Furthermore, the addition and removal of flows between nodes is not supported, which is important for visualizing added and deleted files of a repository.

[1] https://bost.ocks.org/mike/sankey/

The visualization's main feature is the detection of cause and effect of specific code changes. Additional features include a search filter, selecting files and quality metrics, grouping files to modules as well as a panel for details and a legend.

Figure 4.2 shows one of the first sketches of the commit-based view. Nodes represent commits that are linked together, the y-axis represents a selected quality metric and the x-axis represents time. The size of the commit nodes represent the amount of files that were modified by the corresponding commit. We can derive from the commit view that between the commit on 2018-01-07 and 2018-01-14 there was a significant trend upwards, followed by a decreasing trend for the selected quality metric.

In Figure 4.3, the view is switched to the file view by clicking on a commit node in the commit view. The quality trends of all files that were modified by the clicked commit are displayed. It can be seen in Figure 4.3 that the file "options-panel-component.ts" (yellow trend line) has trended up, the files "app-component.html" (pink trend line) and "trend-chart-component.css" (red trend line) have trended down and the file "trend-chart-component.html" (magenta trend line) has spiked up within one commit and has trended down in the next commit. In later versions of the prototype, file trends of the same file but with different quality metrics can be set in relation to each other. With this feature, further statements like "*Test Coverage* has increased with *Cyclomatic Complexity* of this file" are possible.

Figure 4.4 shows the combination of both views where the file view is pushed to the background by adjusting its opacity. The depiction of commits and files in Figure 4.4 allows to set the development of a file's quality trend in relation to a commit's quality trend. This allows to rate how the quality of a file has influenced the quality of its corresponding commit. This can be useful for identifying outliers within a commit's modified files. Note that Figure 4.4 displays other file trends than Figure 4.3.

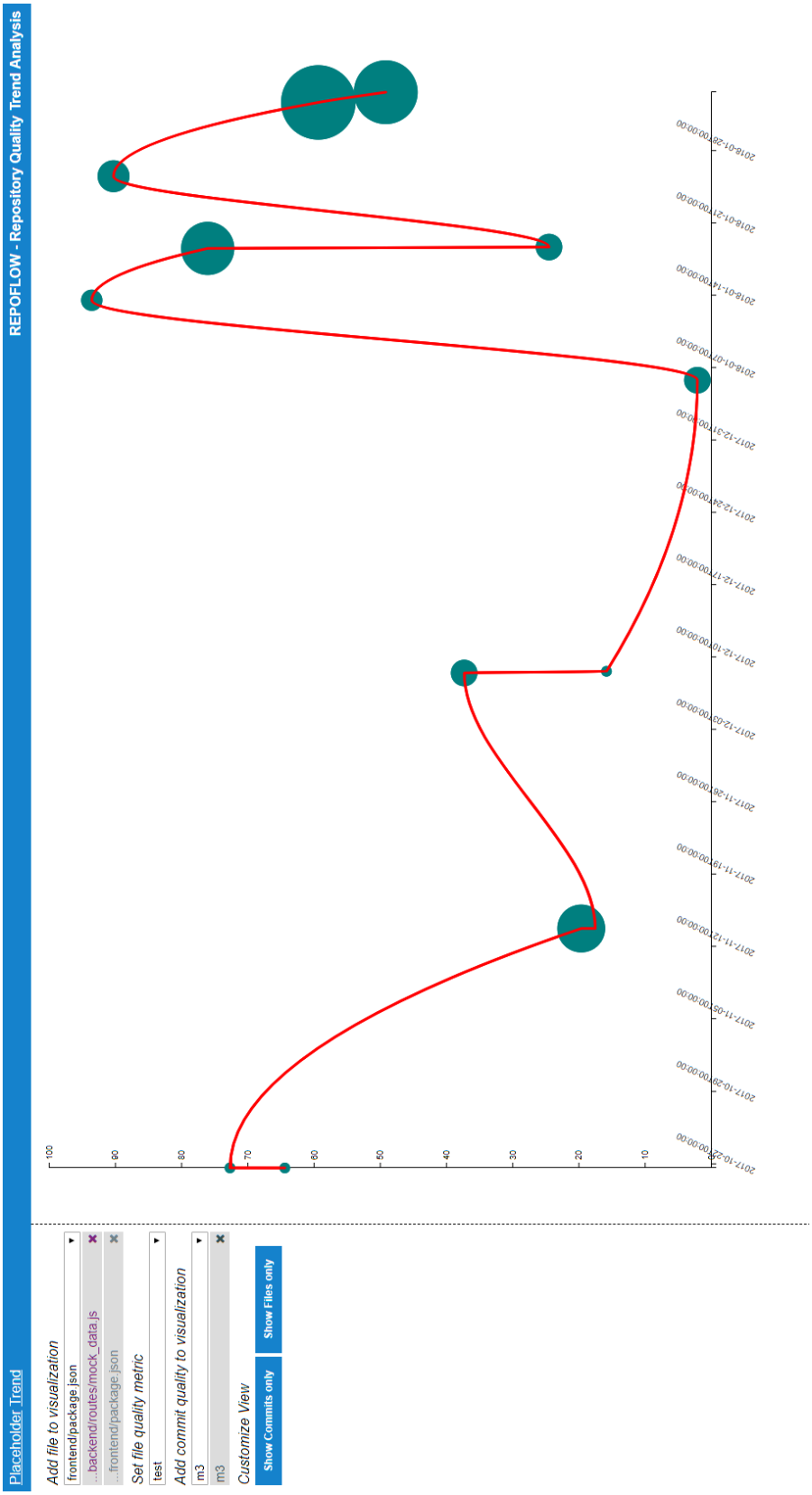


Figure 4.2: Early prototype of commit view.

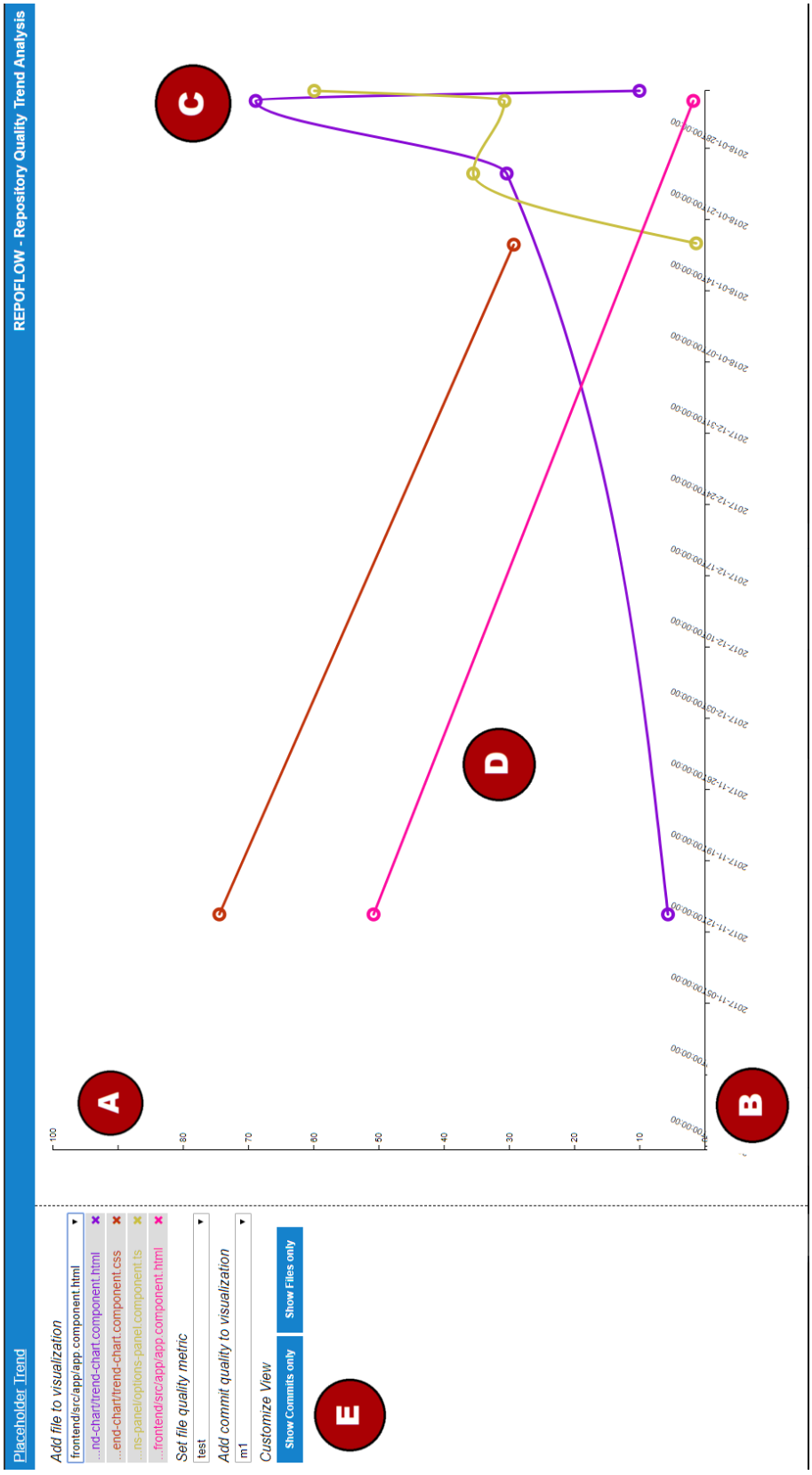


Figure 4.3: Sketch of the visualization method for the file view. A depicts the quality scale whereas B depicts the time scale. C represents a commit that is visualized as a hollow circle, where the file flows through. D shows a line that represents the quality flow of a file from one commit to another. E depicts the filter options area.

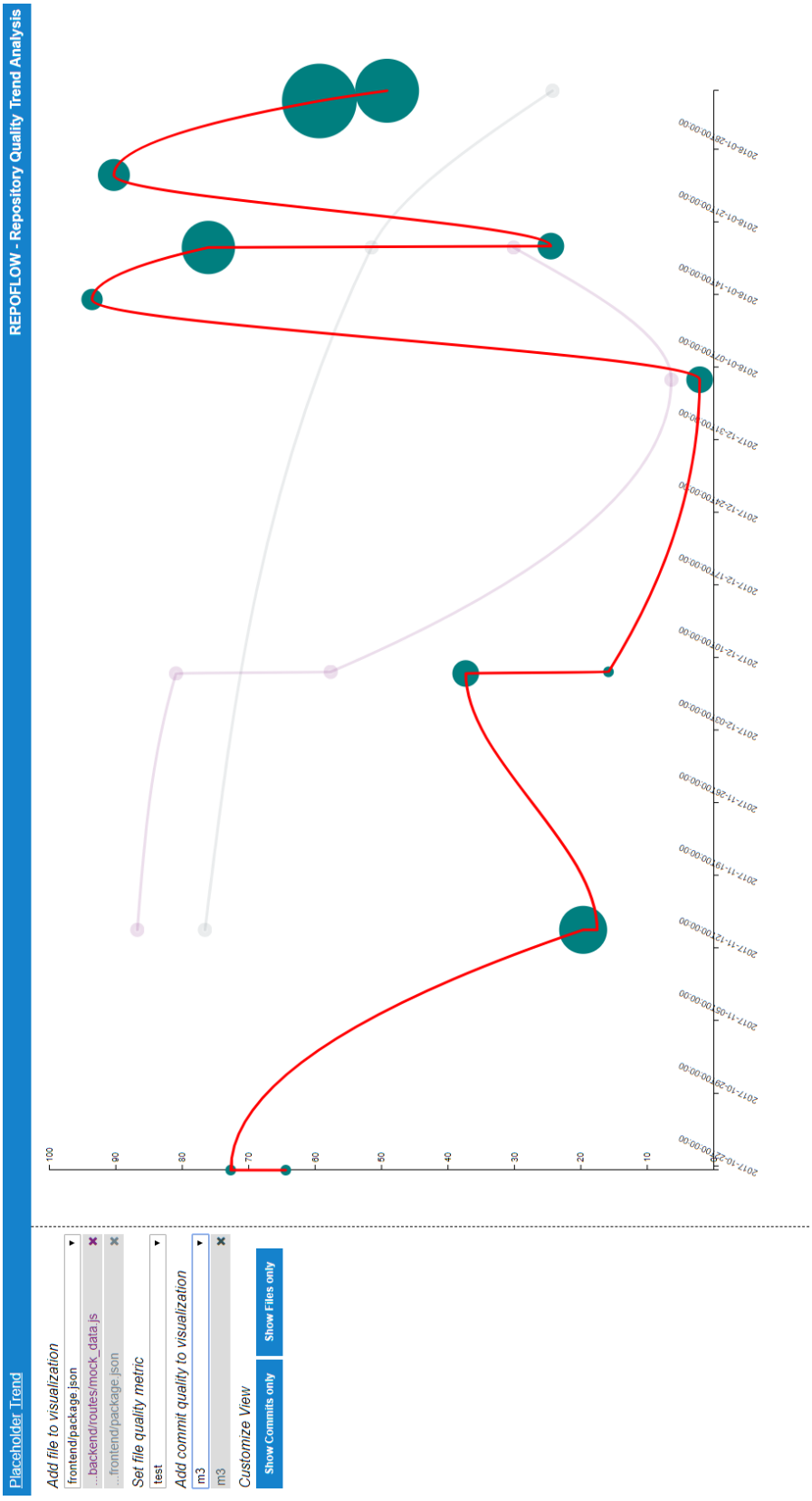


Figure 4.4: Early prototype of combined views.

4.2 Stakeholders

While defining the requirements for the visualization it is important to consider all persons that have an interest in it. These persons are stakeholders.

4.2.1 Identifying Stakeholders

As a first step, it is important to identify all stakeholders and then prioritize them based on the risk they pose to the visualizations' quality by ignoring or neglecting their requirements. These stakeholders are also critical for the evaluation of the visualization [24]:

1. Quality Manager
2. Project Manager
3. Software Architect
4. Developer

Developers and Software Architects share similar requirements for software visualizations and are therefore considered as one group. The following sections describe the requirements of the identified stakeholders based on literature studies for high developer needs in software visualization [13], [7].

4.2.2 Quality Manager

A quality manager is responsible for monitoring software during the development process. They focus on adhering to software standards, ensuring code quality and that the shipped product is released according to schedule. Quality Managers share some requirements with project managers:

- Monitoring: understand the dynamics within the development cycle. For example, what happens if a new developer joins the team?
- Predicted defect density: recognize possible future trends for specific metrics in the visualization's quality data.
- Decide when a feature or code snippet is good enough for release: based on the visualization's data, decide which parts of the project can be released

4.2.3 Project Manager

Project managers monitor and guide the work of designers, developers, and testers of software while sometimes participating in these activities themselves. Project managers focus on high-level concerns like the direction of the project, allocation of resources, feature set and user experience [13]:

- Anticipate changes: recognize possible future trends for specific metrics in the visualization's quality data.
- Risk Management: improve the precision of risk models by interpreting the visualization's quality data.
- Monitoring: understand the dynamics within the development cycle. For example, what happens if a new developer joins the team?

- Improve Efficiency: make informed decisions based on the visualization. For example, map engineers to the tasks they are best at.

4.2.4 Software Architect and Developer

Software architects and developers focus on code architecture and performance [13]. Some of them may have personnel responsibility, shifting their needs and requirements more to that of a project manager:

- Readability of Code: find out which files have bad understandability based on quality metrics.
- Structure of Code: assess the structure of code with different code metrics displayed in the visualization.
- Refactoring: judge quality of code with insights provided by the visualization's data and decide which files must be refactored.
- Impact of code changes: how does a change of the source code influence the development of a file or the commit?

4.3 Implementation of Stakeholders' requirements

Based on the requirements of stakeholders from the previous section, features are defined that implement said requirements. The following Table 4.1 describes features from the visualization and Table 4.2 relates them to the defined requirements from section 4.2. As the requirements of the major stakeholders are critical for the visualization's usefulness, their requirements are prioritized.

Key	Feature
FT1	Utilizing a trend chart for files to understand the development of files and the project as a whole
FT2	Utilizing a trend chart to predict possible future developments within the project
FT3	Commit view with balloon bars to distinguish commits from files
FT4	Y-axis that maps quality on a normalized scale to display different quality metrics (absolute and relative values)
FT5	X-axis for the time scale
FT6	Zooming in and out on the time axis to have a fine-grained visualization - from years to seconds
FT7	Display different files for one quality metric and display different quality metrics for the same file
FT8	Display quality of commits based on the file quality
FT9	Possibility to compare different file versions between two commits
FT10	Search for files from the VCS
FT11	Filter the visualization based on file names, authors, specific files that belong to one specific commit

Table 4.1: List of Features associated with a key.

Requirement	**Implementation**
Anticipate changes: recognize possible future trends for specific metrics in the visualization's quality data.	FT2
Risk Management: improve the precision of risk models with interpreting the visualization's quality data.	FT1 / FT4
Monitoring: understand the dynamics within the development cycle. For example: what happens if a new developer joins the team?	FT11
Improve Efficiency: make informed decisions based on the visualization. For example: map engineers to the tasks they are best at.	FT1 / FT11
Readability of Code: find out which files have bad understandability based on files' metrics.	FT1 / FT10
Structure of Code: assess the structure of code with different code metrics displayed in the visualization.	FT1 / FT9
Refactoring: judge quality of code with insights provided by the visualization's data and decide which files have to be refactored.	FT1 / FT3 / FT4 / FT5 / FT6 / FT7 / FT11
Impact of code changes: how does a change of the source code influence the development of a file or the commit?	FT4 / FT5
Decide when a feature or code snippet is good enough for release: based on the visualization's data, decide which parts of the project can be released	FT1 / FT3 / FT4

Table 4.2: List of requirements related to features that fulfill the requirement.

5 Implementation

This chapter describes how the visualization method was implemented and covers the design choices for implementation, architecture, and API.

5.1 Technology Review

The most important consideration before starting the development of the prototype was that the visualization should be platform-independent. Therefore, the choice was made to build a platform based on web standards and technologies. These technologies and standards include Hypertext Markup Language (HTML), Cascading Stylesheet (CSS), Scalable Vector Graphics (SVG) and JavaScript.

D3.js is purely based on these technologies and standards and was chosen as a framework for the visualization itself.

Angular is a web framework for developing web applications. One of the main reasons to choose Angular is its extensibility that allows creating a library of reusable components. These components can be reused as Angular code can be integrated into any existing web app [29].

Node.js is used to build the API of RepoFlow. Node.js is based on the Google JavaScript engine and allows to build fast, scalable server-side applications. Furthermore, it has built-in support for multiple modules via the Node Package Manager (NPM).

Git is used as a VCS. With Nodegit[1] as a package published in NPM, Git can be mined to retrieve data like version history of a file, commit history, modified lines of code and more. Git is currently the only VCS that is supported by RepoFlow.

ArangoDB is a Not Only Structured Query Language (NoSQL) database. It was chosen to support the exchange of data in JSON format between the frameworks and technologies described above.

5.2 Architectural Overview

RepoFlow consists of three parts that ultimately build the visualization. These parts can also be separated for possible future work. Figure 5.1 shows the basic architecture of RepoFlow, consisting of the API, the database, the Command Line Interface (CLI) and the visualization. Additional metrics can be computed outside of RepoFlow's environment.

The API handles incoming calls and returns values in JSON format. Most calls return entries from the database, for example, a list of files for a specific revision with their corresponding quality metrics. The API also provides calls for external tools to enable computation of custom quality metrics.

The CLI calls the API's methods and helps with creating the database from a fetched repository.

[1] https://github.com/nodegit/nodegit

The visualization also calls the API's methods and renders the retrieved values into the visualization. Also, the visualization handles the User Interface for filtering, tooltips, the editor in the difference view and many other elements.

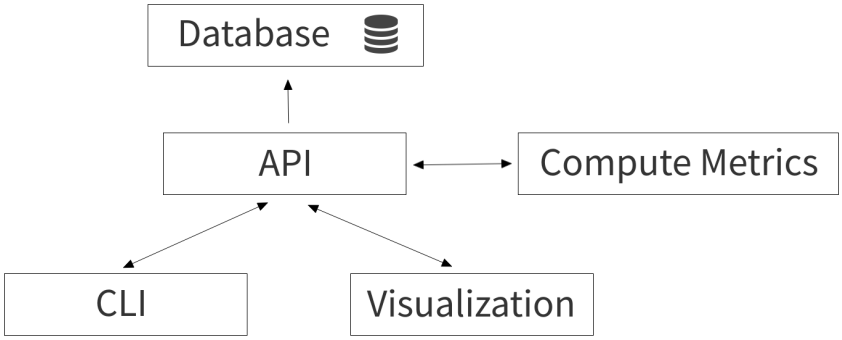


Figure 5.1: Basic architecture of RepoFlow. The database is used to store entities like files or commits. The API serves all essential methods for saving and retrieving values either in the Command Line Interface or the Visualization. The API hosts basic quality metrics for JavaScript. Custom quality metrics can be computed outside of RepoFlow's JavaScript environment.

5.3 Database

Data is stored persistently in a database that uses a NoSQL database. To support storing and reading of JSON objects easily, ArangoDB is chosen. ArangoDB enables to store JSON objects and query the database for their properties. RepoFlow has a total of four collections.

5.3.1 Commit

One of the collections is the commit collection. It holds information like commit author, the commit's Secure Hash Algorithm (SHA) or date and time about a single commit. Quality metric values, as well as quality metric keys, are stored in a dynamic way to each commit. This means that additional JSON properties and values for quality metrics only get added to a commit entry if the API receives a valid call to store these properties and values to the database.

5.3.2 Files

The file collection stores every version of a file that can be retrieved by Git. Every file entry is linked to a commit with the commit's SHA. The stored properties are the filename, the commit's SHA the file belongs to, the file version's status and the file content. Quality metric values, as well as quality metric keys, are stored in a dynamic way to each file version. This means that additional JSON properties and values for quality metrics only get added to a file entry if the API receives a valid call to store these properties and values to the database.

5.3.3 File Color

The file color collection ensures that every file within a repository has a different color coding. Therefore, a color from a color table with distinct colors is assigned to each file.

5.3.4 Quality Metric

Stores every quality metric that is currently available. Every entry stores the available file types for a quality metric, its key as well as a label in an array. Currently the prototype supports five quality metrics for JavaScript by default. These quality metrics are *Cyclomatic Complexity*, *Comment Lines*, *Parameters*, *Statements* and *Lines of Code*. These quality metrics were chosen, as their computation is relatively easy compared to other quality metrics. The computation of advanced quality metrics is not within the scope of this master's thesis. Furthermore, these quality metrics can be applied on a file-level.

5.4 API

The API is based on Express[2], a web framework for Node.js. The API handles incoming calls from the CLI as well as the visualization itself. It returns values as JSON objects.

The API also hosts a configuration file where all relevant information for the database connection can be set.

5.4.1 GET Calls

The API provides calls to retrieve values that are processed by the front-end and visualized in the trend chart area.

get/initial_data returns all commits, files, file colors and quality metrics stored in the database as arrays within a JSON object.

get/commit_data returns all commits stored in the database as an array within a JSON object.

get/file_data_by_name/:filename returns all revisions of a file with the corresponding commit data based on the filename as a JSON object.

get/file_data_by_sha/:sha/:quality_metric_key returns all files of a specific commit by SHA and the filetype that is stored for the given quality metric.

get/files_with_sha returns content, name, SHA, and database key of all files of all revisions. As *npm*[3] does not offer computations for quality metrics for all given languages, this call enables retrieval of all file revisions to be able to compute quality metrics in other environments. In the next section 5.4.2, the call `post/quality` enables to save values that were computed elsewhere as long as the given JSON object adheres to a specific format.

get/min_max_for_metric/:metric returns the minimum and maximum value of a specific quality metric in all files.

get/file_data_by_quality_metric_key/:metric returns all files that match the stored filetype for the given quality metric.

get/clone clones a repository to a given directory. The Uniform Resource Locator (URL) of the repository as well as the directory have to be in the request body.

get/normalization_value returns the minimum and maximum values of all quality metric values.

[2] http://expressjs.com
[3] https://www.npmjs.com/

5.4.2 POST Calls

The API provides calls to store values in the back-end. Especially, custom computed quality metrics outside of RepoFlow's environment can be stored or modified with these calls.

post/database Based on a given repository in the body's request, this call creates the database for the repository. Nodegit is an NPM package and handles the extraction of data from the repository. First commits are extracted from the repository and then inserted into the database with date, message, author and SHA of all commits. Next, modified and created files are extracted and inserted for each commit. Following, each file version is updated with individual colors and the content from each revision. This call returns a JSON object with a status number and a message. Colors that are used are stored in an array in the API. During the creation of the database, these colors are used for different files. As it is not easy to generate as many distinct colors as there are files in a repository via an algorithm, a fixed color array of approximately 1024 different colors is used. This is the most reliable way to ensure that colors are distinct within the visualization, however if the file count of a repository exceeds 1024, it is possible that two files share the same color within the visualization.

post/truncate This call clears all data from the database and returns a JSON object with a status number and a message.

post/quality This call either creates default JavaScript metrics or a custom metric and interacts with `get/files_with_sha`. If the parameter `compute_js_metrics` in the body's request is 1, the call computes default JavaScript metrics for all .js files. If the parameter `files_with_sha_and_computed_metric` is not empty or undefined, the data of this parameter is used to create a custom created quality metric. The parameter `files_with_sha_and_computed_metric` has to be of the following format:

```
[
  [
   { key:qualityMetricKey, label:qualityMetricName,
  ↪ file_types:['.js',...] }
  ],
  [
   { _key:arangoDBKey,qualityMetricKey:qualityMetricValue,
     anotherQualityMetricKey:anotherQualityMetricValue,...}
  ]
]
```

Listing 5.1: Array with JSON objects to define custom metrics.

5.4.3 Quality Computation

To be able to visualize a quality metric in the visualization, the quality metric's value, as well as its key, has to be stored in RepoFlow's database for each file and each commit. Therefore, the API provides the call `post/quality`, which is explained in detail in section 5.4.2.

5.5 Command Line Interface

The CLI exposes different methods of the API to enable setting up RepoFlow quickly, for example, cloning a repository or creating the database. The CLI can be linked as an *npm* package[4] called repoflow. Simply calling `repoflow` in the command line followed by a command executes different API calls. If only `repoflow` is called in the command line, a list of available commands is displayed. The CLI is built with commander[5] and uses chalk[6] and figlet[7] for font modifications in command line tools and inquirer[8] for asking questions via a command line prompt.

5.5.1 Clone Repository

Using `repoflow -clone-repository` in the command line calls `post/clone` in the API. The CLI asks for the repository URL as well as the directory where the repository should be cloned into. The CLI will then clone all contents from the repository into the given folder.

5.5.2 Create Database

Using `repoflow -create-database` calls `post/database`. The CLI asks for the path where the .git-folder can be found. As the repository has to be on the local machine, this is the same directory in which the repository was cloned into.

5.5.3 Clear Database

Typing `repoflow -clear-database` calls `post/truncate`. In this case, the CLI does not pose any confirmation dialog, the database entries are simply truncated.

5.5.4 Create Demo Database

Typing `repoflow -create-demodatabase` calls `post/quality`. This generates the current default quality metrics for JavaScript files within the project.

5.6 Visualization

The visualization is built with Angular[9] and D3.js[10]. The following sections describe all modules of the visualization. A big advantage of the modularisation of the visualization is, that different parts are easily exchangeable or could be utilized in other projects.

5.6.1 Design Decisions

Grammel et al. [28] investigated what the most common interpretation problems of information visualizations are. These problems include high visual complexity, unfamiliar visualization types, inappropriate scalings, difficulties with the semantics of measurements, inappropriate levels of abstraction as well as readability problems and missing numbers. Therefore, the design of this visualization tries to avoid these problems as best as possible.

[4] https://www.npmjs.com/
[5] https://github.com/tj/commander.js
[6] https://github.com/chalk/chalk
[7] https://github.com/patorjk/figlet.js
[8] https://github.com/SBoudrias/Inquirer.js
[9] https://angular.io/
[10] https://d3js.org/

As already mentioned in previous chapters, RepoFlow describes trends in data. Harris [30] introduces different graphs, one of which is the line graph used to describe trends. According to Harris [30], smooth line curves should be used as opposed to stepped line curves when using multiple data series, as stepped line curves can be confusing if they intersect with each other.

Data Type

RepoFlow handles time series data in the form of quality metrics of files and commits at specific points in time. For the visualization, we are interested in the relative changes between these quality metric values to make statements like "quality metric A increased between March 2018 and May 2018" or "quality metric B decreased compared to quality metric C". According to Heer et al. [32] as well as Khan et al. [36], line charts are the best fit for this kind of data type.

Visual Encoding

It is also important to consider all visual encodings that are employed in a visualization [32]. RepoFlow uses a two-dimensional area, where the position of a node represents time on the x coordinate and a quality metrics value on the y coordinate. The shapes of the nodes distinguish between a commit and a file. Commits are rectangular while files are depicted as circles. Color encoding is used to differentiate quality metrics between commits. Also, files are distinguished by their color encoding.

Interaction Techniques

Yi et al. [71] defined seven categories of interaction within an information visualization. Table 5.1 lists these seven categories.

Category	Description
Select	Mark something as interesting
Explore	Show me something else
Reconfigure	Show me a different arrangement
Encode	Show me a different representation
Abstract/Elaborate	Show me more or less detail
Filter	Show me something conditionally
Connect	Show me related items

Table 5.1: Seven categories of interaction in information visualization [71].

RepoFlow uses all of these categories while interacting with the visualization. For example, the category Filter is included but not limited to the file list. Each file trend can be shown depending on the condition of the checkbox that hides or shows the file trend.

Visualization Structure

The overall structure of RepoFlow is an important aspect, as it visualizes different layers of abstraction, the commit layer, the file layer but also a code difference view. Therefore, Shneiderman's "Visual Information Seeking Mantra" is employed. This mantra reads as follows [62]:

Overview first, zoom and filter, then details-on-demand.

This mantra can be seen as a basic principle for building information visualizations. The “Visual Information Seeking Mantra” is mirrored in the design of the visualization. First, an overview of commit trends is displayed. The user can then zoom in on the time axis. Clicking on a commit filters the data and provides details on demand of the next layer of the visualization, the file layer. The file layer can be further filtered by selecting specific file versions, consequently displaying details on demand in the form of a source code difference view.

5.6.2 Trend Chart

The trend chart is the core component of RepoFlow. This component is responsible for rendering all elements in the chart area. D3.js is heavily utilized in this module. All service classes for communicating between modules from the shared module (section 5.6.10) are used in the trend chart module to be able to reflect user input in the chart area. The trend chart component uses the components `<app-options-panel>`, `<app-diff-panel>` and `<app-legend>`.

To be able to properly display every quality metric within its full range of values, feature scaling is applied to the coordinates on the y-axis [2]. The y-axis shows a range of values from 0 to 100. The applied feature scaling maps all quality metric values within the range of [0,1] to be able to properly render all nodes in the y-axis domain. The used equation is as follows:

$$x' = \frac{x}{max(x)} \tag{5.1}$$

This equation is used for every computed value of a specific metric to retrieve its mapped value within the range of [0,1]. Figures 5.2, 5.3, 5.4, 5.5 and 5.6 represent different screenshots of the trend chart area, with the most relevant elements of the visualization.

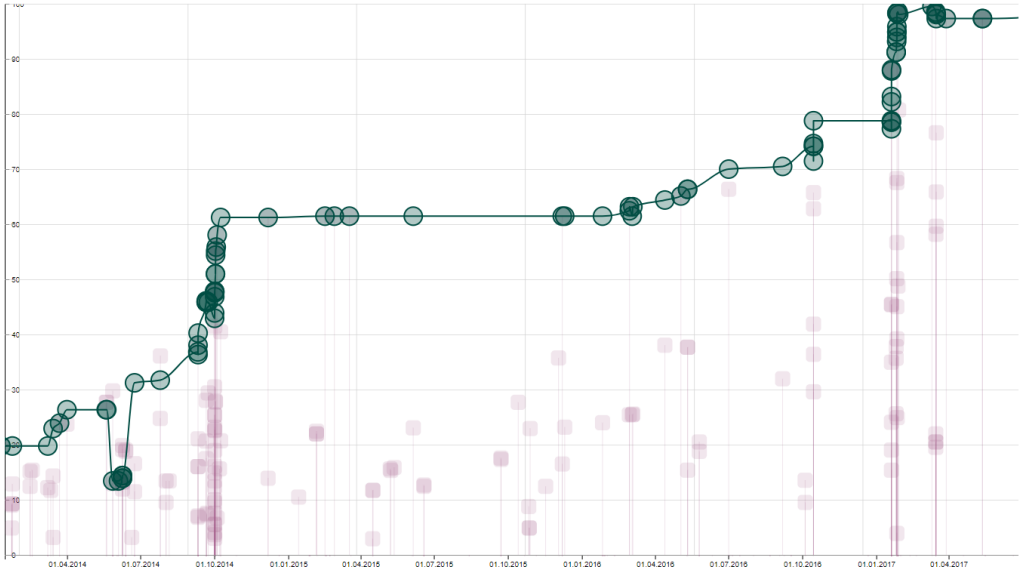


Figure 5.2: Example state of the trend chart area with files. The green trend line represents the trend of a single file for the quality metric *Lines of Code*. The nodes in the background represent commits with the quality metric *Lines of Code*. An interpretation could be as follows: the visualized file (green trend line) has a more than average contribution to the project’s lines of code. This is because the commit nodes in the background represent the average amount of lines of code for each commit and the green trend line for the file is above most of these commit nodes.

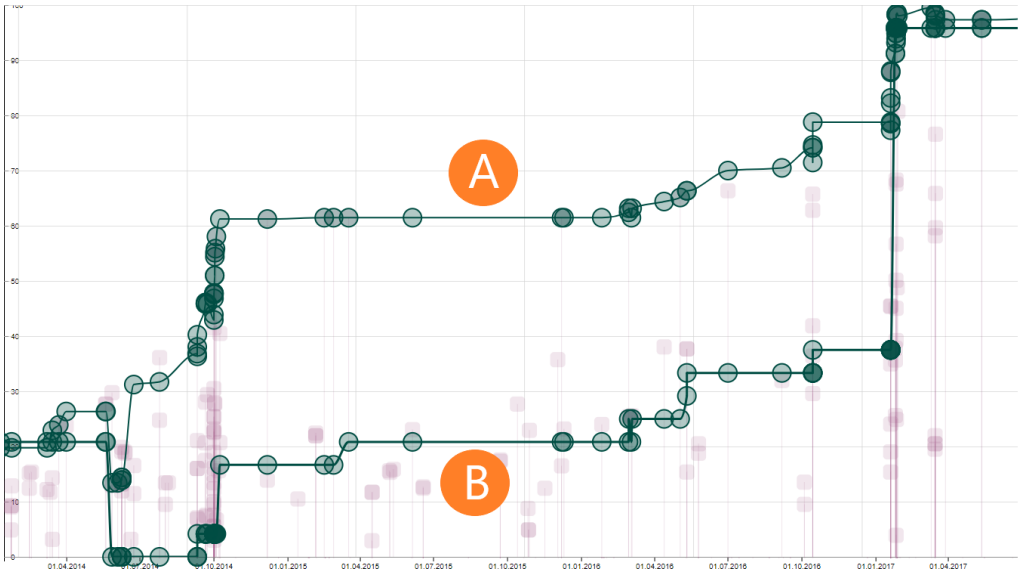


Figure 5.3: Example state of the trend chart area with files. In this screenshot, two quality metric trends are displayed and can be related to each other for further interpretation. The green trend line at (A) represents the trend for the quality metric *Lines of Code*. The green trend line at (B) represents the trend for the quality metric *Comment Lines*. If one of the trend lines is hovered with the mouse, this trend line is highlighted so they can be distinguished from each other. An interpretation could be as follows: first, the number of *Lines of Code* has increased, while the number of *Comment Lines* has decreased, indicating that developers have significantly improved documentation of the code. The nodes in the background represent commits with the quality metric *Lines of Code*.

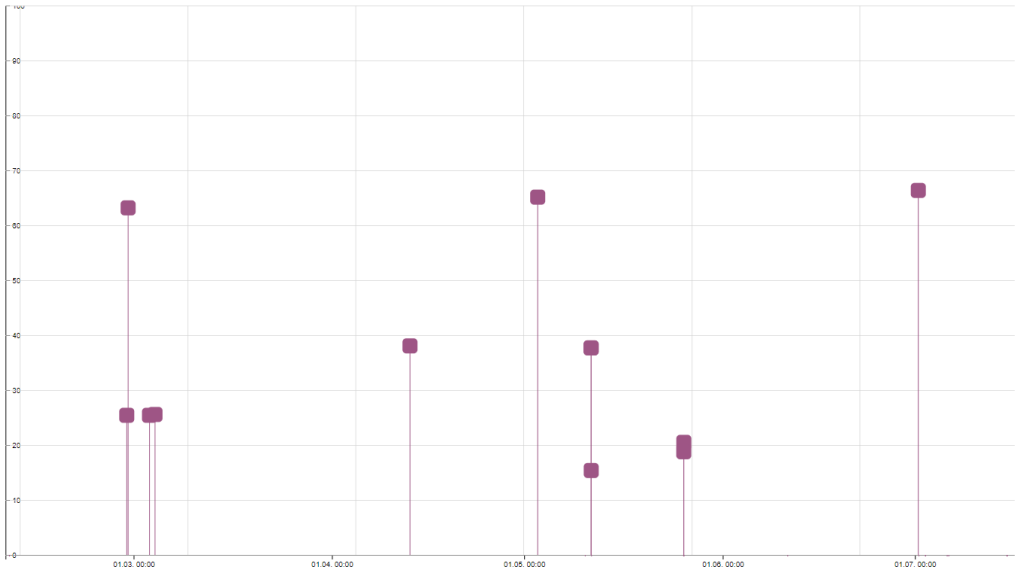


Figure 5.4: Example state of the trend chart area with commits. Only commits with the quality metric *Lines of Code* are visualized in this screenshot.

Figure 5.5: Example state of the trend chart area with commits. Two quality metrics for commits are displayed to be able to compare them to each other. The metric *Lines of Code* is magenta and the metric *Comment Lines* is light green. One possible interpretation of the chart could be that the ratio of *Comment Lines* to *Lines of Code* has decreased over time.

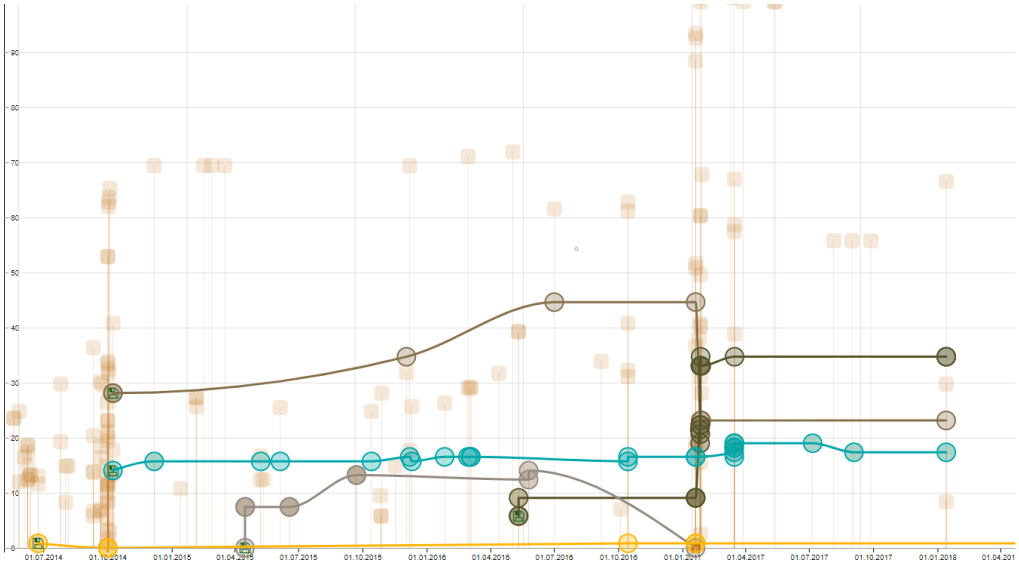


Figure 5.6: Example state of the trend chart area. Different file trends for the quality metric *Cyclomatic Complexity* are visualized. The colored trends in the foreground represent files. The nodes in the background represent commits. An interpretation could be as follows: the trend for *Cyclomatic Complexity* of some of the visualized files shows a notable spike between 01.01.2017 and 01.04.2017. This could be a hint for a larger code refactoring during the time period where all the visualized files were included.

5.6.3 Code Editor

This component is responsible for rendering the code editor in the difference panel. CodeMirror[11] is used for this module to render both editor elements. The `DiffPanelValueService` class is used to exchange the values of the left and the right editor based on user input in the chart area and the difference panel. The code editor also displays the respective quality metric values of the selected file versions above the left and right code editor, so that the user can read out the values directly and is not forced to switch back to the visualization. Figure 5.8 shows the code editor within the difference view of RepoFlow.

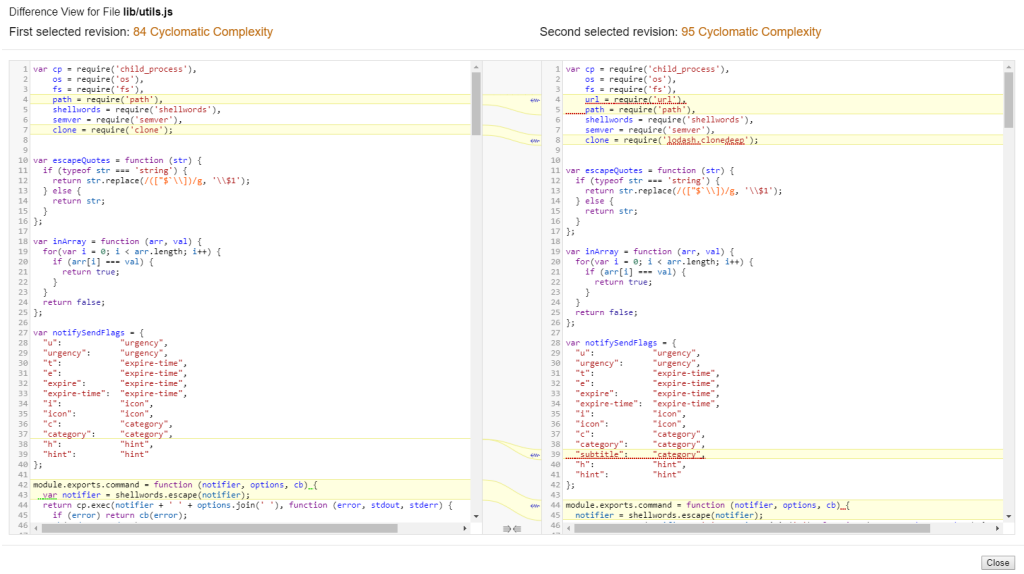


Figure 5.7: Example state of the code editor within the difference view. Two file versions are compared to each other, their respective quality metric values are displayed above the files.

5.6.4 Diff Panel

The diff panel component handles user input that defines which files should be compared in the difference view. This panel also provides a button to open the difference dialog where the code editor is displayed. The HTML uses `<app-code-editor>` from 5.6.3 to render the code editors within the modal dialog. Figure 5.8 shows this component in an example state.

Figure 5.8: Example state of the diff panel within the options panel. Two file versions are selected. Clicking on the button 'Show File Difference View' opens a modal dialog with the code editor and the difference view.

[11] https://codemirror.net/

5.6.5 Group Panel

Files can be moved from the file list to the group panel via drag and drop. This functionality allows to group files to modules. For example, if a commit has multiple files from back-end and front-end, the user may be interested in analyzing the back-end only. Therefore, files from the back-end can be selected in the file list and moved to the group panel via drag and drop. Following, the average value of the selected files from the back-end are visualized in the trend chart area without the values from the front-end. Figure 5.9 shows an example where two files are grouped to a module.

Figure 5.9: The screenshot above shows two files that are selected and displayed in the file list. Moving the files to the group panel via drag and drop in the screenshot below computes the average of the quality values, *Cyclomatic Complexity* in this case, of the two files. Note that the computation of average values only applies to commits that have modified both files.

5.6.6 Legend

This component renders a legend. Currently, it is used by the trend chart to render the legend right under the trend chart area. It can be used with `<app-legend>`. Figure 5.10 shows an example state of the legend with five quality metrics. On the left side of the legend, the nodes of the visulization are introduced via their shapes. A rectangular node represents a commit and a circle represents a file. The colored rectangular shapes represent the quality metrics for the commit nodes. For example, a red rectangular node in Figure 5.10 represents a commit node with the quality metric *Statements*. Every quality metric that is stored in the database is automatically rendered as a colored rectangular node in the legend, therefore the legend is always static. On the right side of the legend, two icons are shown that represent the deletion or addition of a file from respectively to the VCS. These icons are always displayed within a file node in the visualization that is added or deleted from respectively to the VCS.

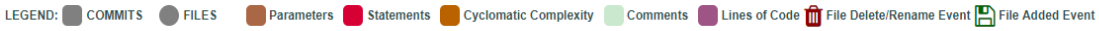


Figure 5.10: Example state of the legend component with five quality metrics stored in the database. The legend is static and always displays every quality metric that is stored in the database.

5.6.7 Modal Dialog

The modal dialog component can be used to render a modal dialog with modular content. Using `<app-modal-header>`, `<app-modal-body>`, `<app-modal-footer>` within an `<app-modal-dialog>` tag can be used to set custom content within a modal dialog. This component is used by the code editor, as well as for tooltips within the options panel

5.6.8 Options Panel

The options panel component is responsible for handling user input in the options panel. It communicates with the `OptionsPanelValueService` class to exchange values with the trend chart. The options panel component uses the `<app-diff-panel>` component. Figure 5.11 shows an example state of the options panel.

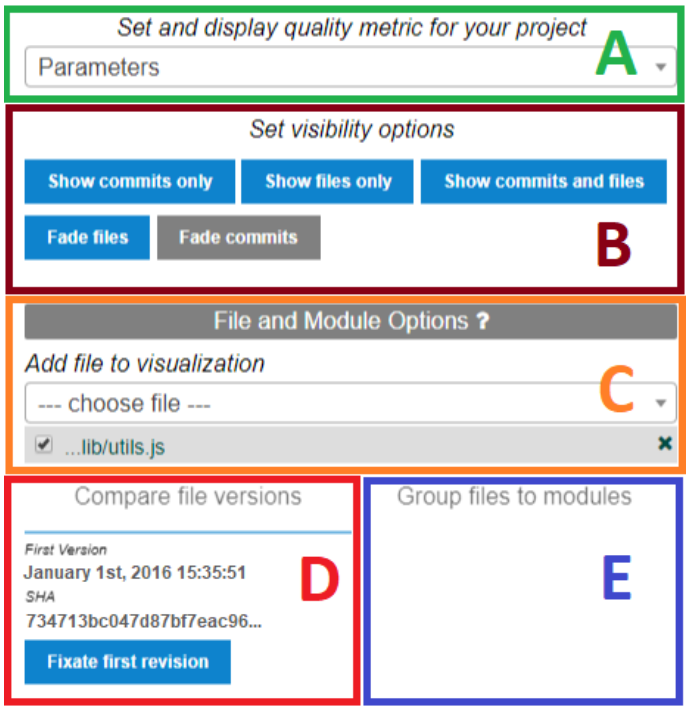


Figure 5.11: Example state of the options panel component. (A) Set the displayed quality metric for commits and files. (B) Set the visibility options for the current visualization in the trend chart area. (C) Select a file to visualize in the trend chart area. In the file list, the file trend can be set to hidden with a checkbox or deleted completely from the visualization by clicking on the 'x' icon. (D) Compare file versions with each other by selecting the versions in the trend chart area. (E) Switching to the 'Group files to modules' tab to drag and drop files from the file list to group these files to modules

5.6.9 Stats Panel

The stats panel displays data for currently focused elements within the visualization. These elements include commit nodes, file nodes and module nodes. For example, if a metric is selected and the mouse cursor hovers over a file, then the value of the metric, the name of the file, the SHA of the file versions commit as well as the commit's date and time get displayed. Figure 5.12 shows an example state of the stats panel.

Figure 5.12: Example state of the stats panel component.

5.6.10 Shared

The following components are service classes that help components to set and retrieve values from each other or from the back-end.

api.service.ts The `ApiService` class is responsible for calling the API's web services. All methods of the ApiService class return JSON objects. All components use methods of this class to retrieve values from the API, respectively the database.

diff-panel-values.service.ts The `DiffPanelValuesService` is responsible for setting and retrieving values that are needed in the diff panel or the code editor according to the user input from the chart area.

options-panel-values.service.ts The `OptionsPanelValueService` class is responsible for setting and retrieving values to and from the options panel component based on user input.

utility.service.ts The `UtilityService` class exposes different methods, for example returning a filename to color lookup array or a datetime comparator.

5.7 Limitations

This section describes the limitations of the visualization's prototype. The limitations are mostly due to the scope and time restrictions of this master's thesis.

5.7.1 Branches

It is currently not possible to switch branches within the visualization. It would be beneficial to be able to compare the trending of quality metrics of files as well as commits between different branches.

5.7.2 Switching between different projects

The CLI is currently the only way to switch between different projects. All data from previous projects are truncated during the process of creating a new project with the CLI.

Another possible way to switch between projects would be a dropdown element that holds all projects stored in the database. Additionally, every database entity needs a project ID.

5.7.3 Number of Quality Metrics

As the computation of quality metrics is not within the scope of this master's thesis, only a few selected quality metrics are available. However, it is possible to compute quality metrics outside of RepoFlow's environment and set the values with the API.

6 Evaluation

RepoFlow's visualization is evaluated with a scenario-based expert evaluation. The general structure of the evaluation is based on Nielsen's approach [53]. First test goals and a test plan are defined. After that, tasks are formulated. To further bolster the practical relevance of this prototype, each participant of the expert evaluation rates every task in regard to its practical relevance in a real-world environment.

This expert evaluation focuses on the utility aspect of the developed prototype. However, the author of this thesis also employs a SUS to assess the usability of the developed prototype [10].

6.1 Goals

The main goals of the expert evaluation are to gather insights into the utility aspect and possible improvements of RepoFlow's visualization.

1. Identify the weaknesses of the user interface and interaction behavior within the visualization and see how these weaknesses can be improved.
2. Assess the implemented features that are based on the feature stack defined in section 4.3.
3. Assess the practical relevance of the prototype with a rating of the employed tasks
4. Gather data about possible improvements to any part of the user interface or visualization.

Deriving from these goals, the conducted evaluation is a mix of qualitative and quantitative evaluation methods.

6.2 Test Plan

To be able to lay out the process of the expert evaluation, a test plan is created according to [53]. A single test session is expected to take 60 minutes, including the completion of ten tasks as well as a demographic questionnaire, a task rating and a SUS questionnaire. A desktop computer or notebook with a resolution of 1080p and Windows 10 is required. An instance of RepoFlow with a test project is pre-installed. The visualization will be in its initial state at the start of each task. The expert evaluation involves 10 participants. Furthermore, two test sessions for the expert evaluation will be conducted. Every participant will be asked to perform a set of tasks that target the utility aspect of the visualization. Each task will define fulfillment criteria that have to be met so that the task counts as correctly finished. A short printed manual of the visualization will be made available to each participant.

6.3 Research Questions

As research questions describe what is expected to learn from the expert evaluation, it is important that these questions are precise, clear and measurable or at least observable [58].

- RQ 1 Is the developed prototype useful for inspecting code quality of a source code repository?
- RQ 2 Does the prototype enable users to express why a project's quality has developed into a specific direction?
- RQ 3 Is the prototype of practical relevance for stakeholders that are interested in quality metrics?

6.4 Method

First, each participant conducts a demographic questionnaire. The demographic questionnaire focusses on the participant's experience with software engineering in general, software repositories as well as software quality metrics.

Then, each participant gets a quick start guide, see Figure A.1, in the form of a printed manual of RepoFlow and executes ten tasks from the task list. The tasks are the same for each participant, but they are not executed in the same order to account for learning effects. The order of the tasks is defined by the Latin square design [59], which will be explained in detail in section 6.7.1. The tasks were chosen in a way so that all major features of RepoFlow are used by each participant.

To ensure the validity of the expert evaluation, it is important to select the right users as well as the right tasks [53]. All participants have a software engineering background and use repositories on a regular basis and have at least basic knowledge of software quality metrics.

The participant is asked to think aloud during the execution of the tasks. This evaluation method may slow users down or influence their problem-solving behavior during the session as they have to verbalize their thoughts [53]. However, it is crucial to get insight into the thought process of users while they are using the visualization, as this may identify what hinders them to complete the given task. Therefore, the focus of the expert evaluation is not primarily the time of execution but instead the successful completion criteria for each task.

With the participant's consent, audio and screen of the session are recorded for easier evaluation and reproducibility of the results.

After finishing the tasks, each participant rates the previously executed tasks based on their relevance in a real-world environment.

Each participant concludes the expert evaluation with a follow-up interview where the participant is asked for general feedback and an SUS questionnaire. The interview aims at finding out, which aspects of the RepoFlow visualization need improvement based on the participant's experience.

6.5 Introduction Protocol

To properly introduce each participant to the expert evaluation, an introduction protocol is set up. Before the participant begins with the evaluation, the interviewer introduces the general procedure based on the introduction protocol. The interview protocol also makes sure that the interviewee gets all important information before the interview [34]. The text below is the introduction protocol for RepoFlow's expert evaluation. Instructions for the participants are described in square brackets.

> Thank you for taking part in this expert evaluation. You evaluate RepoFlow, a prototypical visualization of software quality trends of files and commits gathered from a software repository.

You will stop at four stations beginning with a demographic questionnaire.

[Point to the first tab in the browser window.]

After that, you will execute ten tasks that can be solved with the prototype. Please solve the tasks on your own, but if you have any question regarding the wording of a task, feel free to ask. Please 'think aloud' during the execution of a task and indicate when you have finished a task. Tasks will be handed to you one after another on a sheet of paper. Please refresh the visualization after each task by pressing F5 to set the visualization to its initial state.

The visualization is already set up with a real-world example and accessible through the browser.

[Point to the second tab in the browser window.]

After the test, we will conduct a rating of each task regarding its practical use. The rating is open in the third tab. We provide an example of a practical use case for each task, however, if you need further scenarios to assess a task, we will provide them, as it should not be your responsibility to find scenarios.

[Point to the third tab in the browser window.]

Following the task rating, we will evaluate the prototype with a system usability scale on a sheet of paper. You are asked to answer 10 questions regarding your experience with the visualization and voice your general opinion about the visualization afterward. We also provide a quick start guide for the visualization to give you an introduction of its features and how they work. Feel free to use the quick start guide during the execution of tasks.

[Hand over the quick start guide.]

Please take some time now to read the quickstart guide before the test session. If you are ready, we will begin with the Demographic Questionnaire.

[Prepare audio and screen recording - start if consent is given during the demographic questionnaire.]

6.6 Demographics

6.6.1 Participants

The test will be conducted with ten participants that have expertise in the field of software engineering. Nielsen [53] states that the maximum benefit-cost ratio for participants is five. However, more recent research shows that the detection rate of problem highly fluctuates with a low number of participants. Faulkner [19] states that a randomly selected set of five participants found 99% of problems while another set only found 55%. With ten participants, the lowest percentage of problems revealed by any set was increased to 80%.

6.6.2 Demographic Questionnaire

Each participant fills out a demographic questionnaire form, see Table A.3 that focuses on the participant's software engineering experience, repository experience as well as software quality experience.

6.7 Task List

According to [53], tasks should be as representative as possible to tasks that will be executed in a real-world environment. Therefore, the selected tasks derive from the feature list defined in Table 4.1, as well as the research questions defined in section 6.3.

To be as representative as possible for a real-world environment, the project 'node-notifier'[1], available via the node package manager, is chosen for evaluation purposes. It has a high download counter (about 4 600 000 between February 12th 2019 and February 18th 2019) and is actively maintained. All tasks are based on the repository data of 'node-notifier'.

Each task defines a description that explains the shortest path to the solution. Furthermore, each task defines a question. The task question is handed over to each participant on a sheet of paper. Each task has a solution and successful completion criteria to be able to measure if a task has been finished correctly. The benchmark for each task is the time it takes to meet the successful completion criteria. The following Tables 6.1 to describe each task in detail.

Description	Select the quality metric Cyclomatic Complexity from the dropdown list on the top left of the options panel.
Question	What is the value of the Cyclomatic Complexity of the commit on January 26, 2016?
Solution	31.83 Cyclomatic Complexity
Successful completion criteria	The participant finds out the correct solution value of 31.83 Cyclomatic Complexity.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.1: Task A: What is the value of the Cyclomatic Complexity of the commit on January 26, 2016?

Description	Select and visualize the file 'lib/terminal-notifier.js' with the quality metric 'Parameter' set.
Question	What is the value of the metric Parameters for the file 'lib/terminal-notifier.js' on January 22nd, 2014, 21:49:11?
Solution	10 Parameters.
Successful completion criteria	The participant finds out the correct solution value of 10 Parameters.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.2: Task B: What is the value of the metric Parameters for the file 'lib/terminal-notifier.js' on January 22nd, 2014, 21:49:11?

[1] https://github.com/mikaelbr/node-notifier

Description	Visualize all files belonging to the commit from July 1st, 2016 12:34:19 and read out the value of the highest trending file.
Question	Which file from the commit on July 1st, 2016 12:34:19 has the highest number of Lines of Code?
Solution	'lib/utils.js'.
Successful completion criteria	The participant finds out the correct file 'lib/utils.js'.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.3: Task C: Which file from the commit on July 1st, 2016 12:34:19 has the highest number of Lines of Code?

Description	Visualize the commits with number of Lines of Code selected. Select the file 'lib/notifiers/terminal-notifier.js'. Fade file nodes to the background to compare the selected file nodes to the commit nodes.
Question	Visualize the file 'lib/notifiers/terminal-notifier.js' with the quality metric Lines of Code set. Is the file below or above the average value of Lines of Code of the commit on July 25th, 2014, 19:15:03?
Solution	Use the faded file nodes to compare the value of the files to the commit value and make the statement that the file is below average.
Successful completion criteria	The participant correctly states that the file's value is below average.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.4: Task D: Visualize the file 'lib/notifiers/terminal-notifier.js' with the quality metric Lines of Code set. Is the file below or above the average value of Lines of Code of the commit on July 25th, 2014, 19:15:03?

Description	Compare the quality trends of Lines of Code and Comments of the file 'lib/utils.js' to each other. Search for the first appearance of 0 in the Comments trend line and read out the quality value of Lines of Code at this point in time.
Question	Relate Lines of Code to Comments for the file 'lib/utils.js' with the context menu. After the file was added, when is the first time that the Comments value is 0 and how many Lines of Code does the file have at this point in time?
Solution	May 27th, 2014 with 56 Lines of Code.
Successful completion criteria	The participant finds out the correct value of 56 Lines of Code and the correct date May 27th, 2014.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.5: Task E: Relate Lines of Code to Comments for the file 'lib/utils.js' with the context menu. After the file was added, when is the first time that the Comments value is 0 and how many Lines of Code does the file have at this point in time?

Description	Select the files 'lib/utils.js' and 'notifiers/balloon.js' on June 5th, 2015, 08:13:32. Switch to the tab Group files to modules and drag and drop both files in the designated area.
Question	Find out the grouped Cyclomatic Complexity value of the files 'lib/utils.js' and 'notifiers/balloon.js' on June 5th, 2015, 08:13:32.
Solution	59 Complexity.
Successful completion criteria	The participant finds out the correct value of 59 Complexity.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.6: Task F: Find out the grouped Cyclomatic Complexity value of the files 'lib/utils.js' and 'notifiers/balloon.js' on June 5th, 2015, 08:13:32.

Description	Visualize the quality trend of the file 'lib/utils.js' with the quality metric 'Lines of Code' set. Hover over the file node on June 5th, 2015 08:13:32 and read out the meta information.
Question	Please name the author and the first ten characters of the commit SHA from the file 'lib/utils.js' on June 5th, 2015 08:13:32 with the quality metric 'Lines of Code' set.
Solution	Name: Mikael Brevik and commit SHA: 9da0d758f0.
Successful completion criteria	The participant finds out the correct name Mikael Brevik and the first ten letters of the SHA 9da0d758f0.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.7: Task G: Please name the author and the first ten characters of the commit SHA from the file 'lib/utils.js' on June 5th, 2015 08:13:32 with the quality metric 'Lines of Code' set.

Description	Visualize the file trend for the file 'lib/utils.js' and choose Lines of Code as a quality metric. Click on the file node on October 3rd, 2014 09:02:40 and click "Fixate First Revision". Then click on the file node on October 4th, 2014 12:59:47 and click "Show File Difference View". Scroll down to the yellow highlighted area where the code change is marked and read out the lines of code.
Question	Which source code change led to the change in quality metric Lines of Code for the file 'lib/utils.js' from October 3rd, 2014 09:02:40 to October 4th, 2014 12:59:47? Find the exact lines of code that are responsible for the change in the quality metric.
Solution	Lines 256 - 267.
Successful completion criteria	The participant finds out the correct value of Lines 256 - 267.
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.8: Task H: Which source code change led to the change in quality metric Lines of Code for the file 'lib/utils.js' from October 3rd, 2014 09:02:40 to October 4th, 2014 12:59:47? Find the exact lines of code that are responsible for the change in the quality metric.

Description	Visualize the quality trend of the file 'lib/utils.js' with Cyclomatic Complexity set. Zoom in until the clustered nodes are all distinguishable and hover over the nodes at the dates in question and read out the values.
Question	What are the quality values of the file 'lib/utils.js' with the Cyclomatic Complexity metric set on the following dates: October 1st, 2014, 12:25:04; October 1st, 2014, 12:28:15;October 1st, 2014; 13:02:22; October 1st, 2014, 13:06:34?
Solution	63, 64, 64, 64
Successful completion criteria	The participant finds out the correct values 63, 64, 64, 64
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.9: Task I: What are the quality values of the file 'lib/utils.js' with the Cyclomatic Complexity metric set on the following dates: October 1st, 2014, 12:25:04; October 1st, 2014, 12:28:15;October 1st, 2014; 13:02:22; October 1st, 2014, 13:06:34?

Description	Visualize the quality trend for Lines of Code for commits. Then use the context menu to additionally visualize the quality trend for Comment Lines. Hovering over the correct date displays the corresponding values.
Question	Relate Lines of Code to Comment Lines with the context menu. What is the commit's value of Lines of Code and what is the commit's value of Comment Lines on May 3rd, 2016, 02:48:32?
Solution	268 Lines of Code and 6 Comment Lines.
Successful completion criteria	The participant finds out the correct values for Lines of Code (268) and Comment Lines (6).
Benchmark	Time it takes to meet the successful completion criteria.

Table 6.10: Task J: Relate Lines of Code to Comment Lines with the context menu. What is the commits value of Lines of Code and what is the commit's value of Comment Lines on May 3rd, 2016, 02:48:32?

6.7.1 Latin-square Task List

As already mentioned, the tasks are arranged in a special order to account for possible learning effects. Table 6.11 represents the order in which each participant executed the list of tasks for the expert evaluation.

	Tasks									
Participant One	C	B	I	F	J	E	G	A	H	D
Participant Two	J	G	H	D	F	I	B	C	E	A
Participant Three	B	J	C	G	E	D	I	F	A	H
Participant Four	G	E	D	H	B	J	A	I	C	F
Participant Five	F	I	G	B	C	A	E	H	D	J
Participant Six	D	F	B	J	A	H	C	G	I	E
Participant Seven	E	D	A	C	H	F	J	B	G	I
Participant Eight	I	H	E	A	G	C	F	D	J	B
Participant Nine	A	C	J	I	D	B	H	E	F	G
Participant Ten	H	A	F	E	I	G	D	J	B	C

Table 6.11: Latin-squared list of tasks for ten participants

6.8 Task Rating

The task rating, see Table A.1, additionally bolsters the practical use of the developed visualization in a real-world environment. Every participant rates each task based on its relevance in a real-world environment. Each task is repeated for the participant and an exemplary practical use case is listed in the questionnaire. The interviewer provides further scenarios for each task if needed.

6.9 SUS Questionnaire

The SUS, see Table A.2 is a simple, ten-item scale giving a global view of subjective assessments of usability [10]. The SUS questionnaire is conducted after the participant completes the given tasks. After the SUS questionnaire, the participant is asked to voice his opinion and give general feedback on the prototype.

6.10 Pre-Test

To verify the employed methods of the expert evaluation, two pre-tests were conducted before the actual evaluation. Participants of these pre-tests also have a software engineering background. They executed exactly the same process as described in the methodology section 6.4. According to Porst [55], the execution of a pre-test is an inevitable requirement for a successful evaluation.

6.11 Results

This section covers the results of the expert evaluation. The results analysis is split into four subsections: Demographics, Task results, Task rating, SUS questionnaire, and general feedback.

6.11.1 Demographics

The average **age** of participants is 32.8 years with a median of 33 years (σ=3.6). Participants' **software engineering experience** is 16.5 years on average with a median of 16.5 years (σ=5.2). The self-assessment of **repository experience** yields an average value of 3.9 with a median of 4 (σ=0.9), on a scale from 0 to 5. Participants' self-assessment of **software quality metric experience** averages out at 2.9 with a median of 3 (σ=1.2) on a scale from 0 to 5.

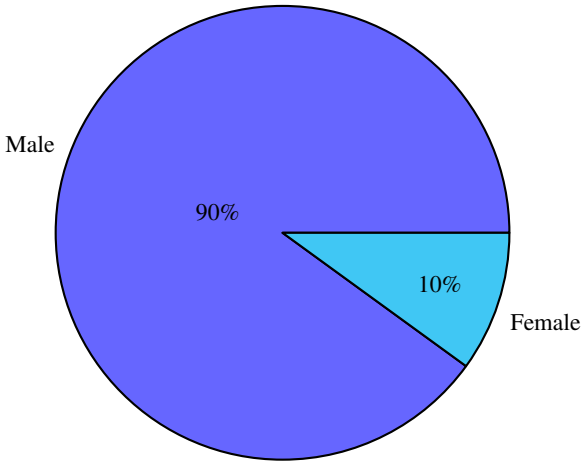
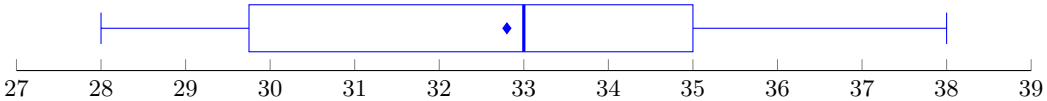
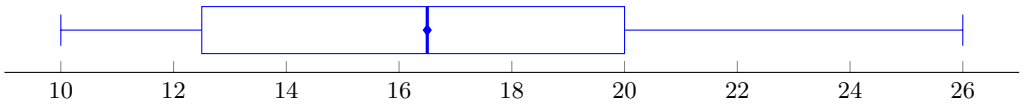


Figure 6.1: Distribution of participants' gender

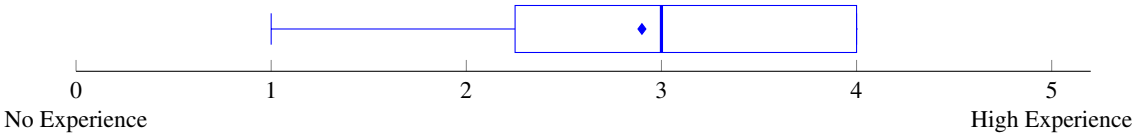


(a) *Participants' age in years (Average: 32.8, Median: 33)*



(b) *Participants' software engineering experience in years (Average: 16.5, Median: 16.5)*

(c) *Participants' self-assessment of their repository expertise (Average: 3.9, Median: 4)*



(d) *Participants' self-assessment of their quality metric expertise (Average: 2.9, Median: 3)*

Figure 6.2: Boxplots for participants' age, software engineering experience, self-assessment of repository expertise and self-assessment of quality metric expertise

6.11.2 Task Results

This section analyzes each task regarding its completion rate, most common problems as well as time taken for completion.

(A) What is the value of the Cyclomatic Complexity metric of the commit on January 26 2016?

- This task has a completion rate of 90%. It took participants 63 seconds on average (σ=21) to complete this task.
- One participant clicked on the commit node of the specified date. The participant expected to see the quality metrics value on click and not on hover. After the click, all

file trends were displayed and the value of the nearest file node instead of the commit node was given as the answer.

(B) What is the value of the Parameters metric for the file 'lib/terminal-notifier.js' on January 22nd, 2014, 21:49:11?

- This task has a completion rate of 100%. It took participants 110 seconds on average (σ=114) to complete this task.

(C) Which file from the commit on July 1st, 2016 12:34:19 has the highest number of Lines of Code?

- This task has a completion rate of 100%. It took participants 101 seconds on average (σ=88) to complete this task.

(D) Visualize the file 'lib/notifiers/terminal-notifier.js' with the quality metric Lines of Code set. Is the file below or above the average value of Lines of Code of the commit on July 25th, 2014, 19:15:03?

- This task has a completion rate of 90%. It took participants 162 seconds on average (σ=64) to complete this task.
- One of the participants clicked on the commit to display all related file trends. The participant then analyzed the file 'test/terminal-notifier.js' instead of 'lib/notifiers/terminal-notifier.js' which was above the average value of Lines of Code.

(E) Relate Lines of Code to Comments for the file 'lib/utils.js' with the context menu. After the file was added, when is the first time that the Comments value is 0 and how many lines of code does the file have at this point in time?

- This task has a completion rate of 80%. It took participants 215 seconds on average (σ=129) to complete this task.
- Some participants initially read out the values when the file was added and not after it was added. The file coincidentally had a comments value of 0 when it was added.
- One of the participants used the difference view to read out the lines of code and gave 67 as the answer. However, the quality metric for Lines of Code only computes actual lines of code and not physical lines with line breaks or comments.

(F) Find out the grouped Cyclomatic Complexity value of the file 'lib/utils.js' and 'notifiers/balloon.js' on June 5th, 2015, 08:13:32.

- This task has a completion rate of 80%. It took participants 200 seconds on average (σ=92) to complete this task.
- A common problem with this feature is the size of the drag and drop area, which is too small. Most of the participants needed at least two tries to drag and drop the file in the area.
- Some participants were confused by the wording of the tab 'Group files to modules' and could not find the drag and drop area.

(G) Please name the author and the first five characters of the commit SHA from the file 'lib/utils.js' on June 5th, 2015 08:13:32 with the quality metric 'Lines of Code' set.

- This task has a completion rate of 100%. It took participants 87 seconds on average (σ=33) to complete this task.

(H) Which source code change led to the change in quality metric Lines of Code for the file 'lib/utils.js' from October 3rd, 2014 09:02:40 to October 4th, 2014 12:59:47? Find the exact lines of code that are responsible for the change in the quality metric.

- This task has a completion rate of 100%. It took participants 244 seconds on average (σ=87) to complete this task.
- Even if this task has a completion rate of 100%, the most common problem here was the naming and the positioning of the button to fixate the two file versions. Most participants' first reaction was to right click on the node and try to find an option in the context menu to fixate or compare the clicked file version to another file version. This task has the highest execution time because participants struggled to find the options they needed to complete the task.

(I) What are the quality values of the file 'lib/utils.js' with the Cyclomatic Complexity metric set on the following dates: October 1st, 2014, 12:25:04; October 1st, 2014, 12:28:15;October 1st, 2014; 13:02:22; October 1st, 2014, 13:06:34?

- This task has a completion rate of 90%. It took participants 151 seconds on average (σ=67) to complete this task.
- One of the participants did not zoom into the visualization far enough to distinguish between overlapping nodes, which resulted in false answer values.

(J) Relate Lines of Code to Comment Lines with the context menu. What is the commits value of Lines of Code and what is the commits value of Comment Lines on May 3rd, 2016, 02:48:32?

- This task has a completion rate of 90%. It took participants 151 seconds on average (σ=86) to complete this task.
- Many participants were irritated by the display of decimals of Lines of Code, as this made no sense on the file trends. One of the participants interpreted 268.00 Lines of Code as 268 000 Lines of Code.

Table 6.12 summarizes the outcome of the evaluation regarding correctness and completion time.

Task	**Correctness Rate**	**Time**
A	90%	63 seconds
B	100%	110 seconds
C	100%	101 seconds
D	90%	162 seconds
E	80%	215 seconds
F	80%	200 seconds
G	100%	87 seconds
H	100%	244 seconds
I	90%	151 seconds
J	90%	151 seconds

Table 6.12: Summary of task results

6.11.3 Task Rating Results

This section shows the results of the task rating. Each task was rated on a scale from 1 to 5. The average rating of all tasks is 3.97.

(A) Exemplary practical use case: analyze quality metrics of specific commits.

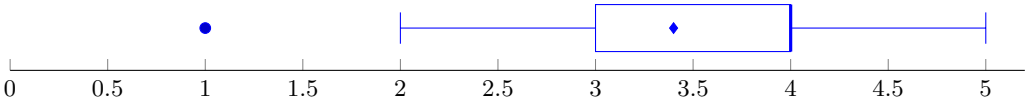


Figure 6.3: Results of rating of Task A

(B) Exemplary practical use case: analyze quality metrics of specific files for specific points in time.

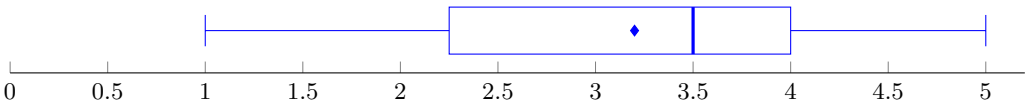


Figure 6.4: Results of rating of Task B

(C) Exemplary practical use case: search for files with highest/lowest quality values within a commit.

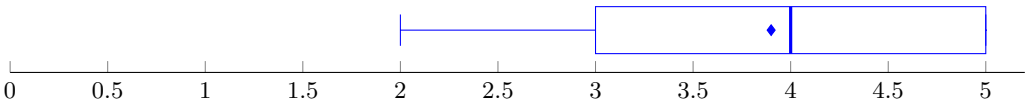


Figure 6.5: Results of rating of Task C

(D) Exemplary practical use case: analyze the impact of single files on the overall quality value of a commit.

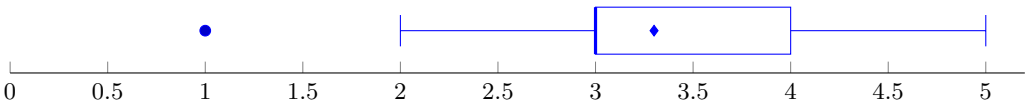


Figure 6.6: Results of rating of Task D

(E) Exemplary practical use case: Compare multiple quality trends of the same file and relate the development of different software quality metrics.

Figure 6.7: Results of rating of Task E

(F) Exemplary use case: analyze the quality value of a module consisting of multiple files.

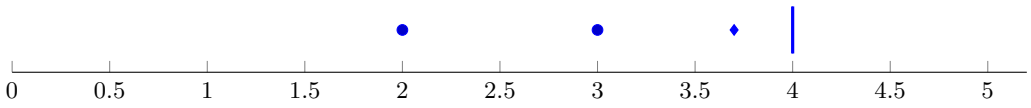


Figure 6.8: Results of rating of Task F

(G) Exemplary use case: read out meta information of specific file versions.

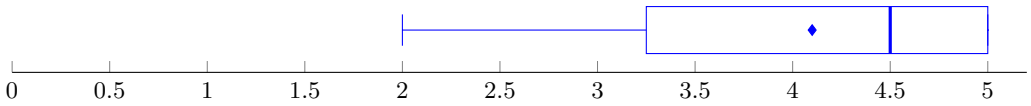


Figure 6.9: Results of rating of Task G

(H) Exemplary use case: find the source code change that is responsible for a change in a quality metrics trend.

Figure 6.10: Results of rating of Task H

(I) Exemplary practical use case: analyze densely clustered files in the visualization (via zooming in on the time axis) and read out their quality values.

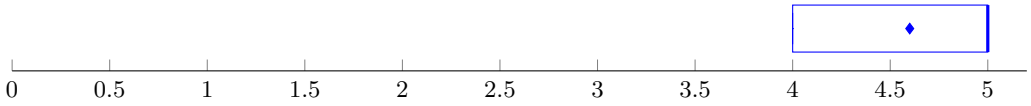


Figure 6.11: Results of rating of Task I

(J) Exemplary practical use case: compare different quality metrics for specific commits.

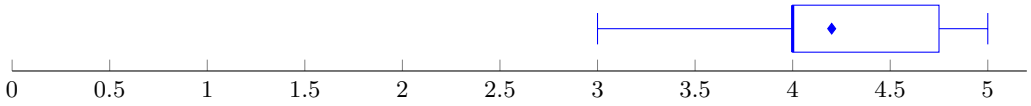


Figure 6.12: Results of rating of Task J

6.11.4 SUS Questionnaire

The SUS questionnaire achieved an average rating of 73.3 (σ=12.5), see Figure 6.13. According to Bangor et al. [6], an average rating of 73.3 can be mapped to an adjective rating of "Good."

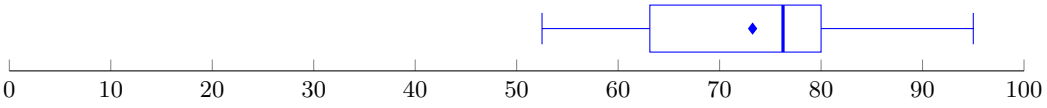


Figure 6.13: Results of SUS questionnaire

The detailed results for each question are as follows:

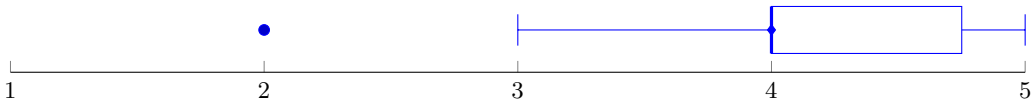


Figure 6.14: I think that I would like to use this website frequently.

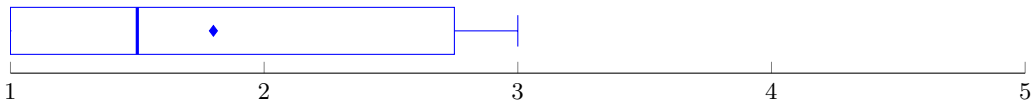


Figure 6.15: I found this website unnecessarily complex.

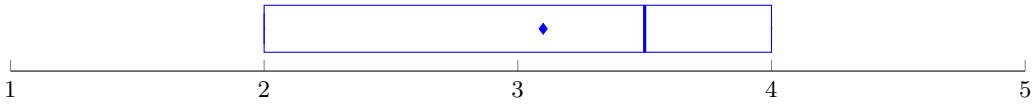


Figure 6.16: I thought this website was easy to use.

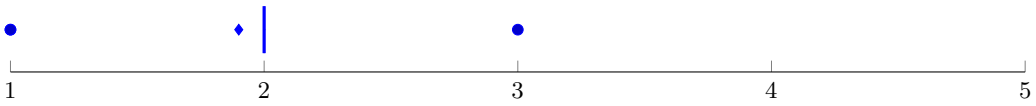


Figure 6.17: I think that I would need assistance to be able to use this website.

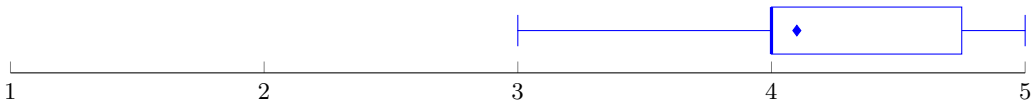


Figure 6.18: I found the various functions in this website were well integrated.

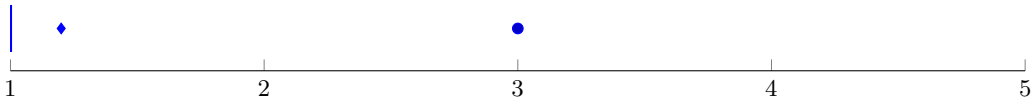


Figure 6.19: I thought there was too much inconsistency in this website.

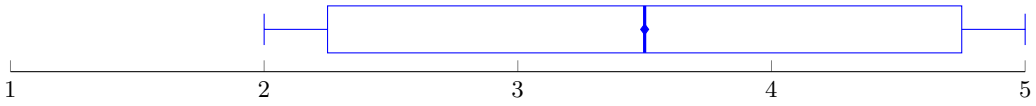


Figure 6.20: I would imagine that most people would learn to use this website very quickly.

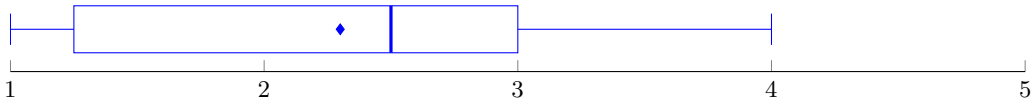


Figure 6.21: I found this website very cumbersome/awkward to use.

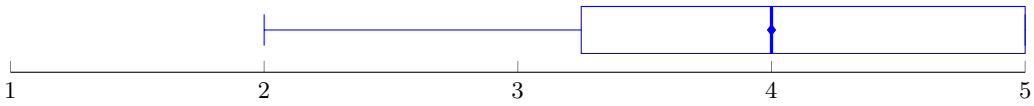


Figure 6.22: I felt very confident using this website.

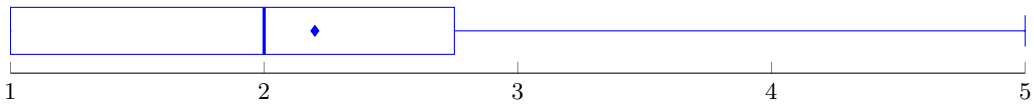


Figure 6.23: I needed to learn a lot of things before I could get going with this website.

6.11.5 General Feedback

Most participants stated that they clearly see an added value of the prototype to existing tools.

The general consensus is that the prototype's visualization is more dynamic than current solutions, as it allows to drill down to a problem cause faster. Especially participants that used respective existing visualizations said that the prototype would make it easier for them to find relevant information.

6.11.6 Problems

As the current implementation is a prototype, several problems unfolded during the expert evaluation.

DateTime Filter Participants want to filter the displayed time range in which they can search for specific values of commit or file nodes.

Time Axis Labeling Zooming into the visualization changed the labeling of the time-axis from 'dd.mm.YY' to 'dd.mm hh:mm'. Participants were irritated by this behavior and had to zoom out again to reproduce which year they are currently visualizing. The problem is illustrated in Figure 6.24.

(a) Display of date in the format 'dd.mm.YY'

(b) Display of date in the format 'dd.mm hh:mm'

Figure 6.24: Time-axis labeling switching between two date formats

Decimals and Decimal Separators As every quality metric always had two decimals, participants often confused the combination of dots and zeros with a thousands separator.

File Version Comparison The buttons for fixating specific file versions to compare them with each other are placed in the options panel. However, participants tried to compare file versions by right-clicking on a file node and expected to find the option in the context menu.

Drag and Drop Area for File Grouping The drag and drop area for grouping files to modules was too small and participants had problems finding the right spot where to drop the files. Figure 6.25 illustrates the area where files can be dropped.

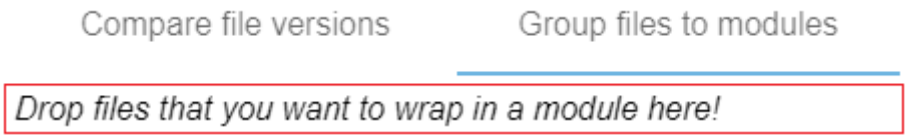


Figure 6.25: Drag and drop area for files to group them to modules. The red rectangle illustrates the area where files can be dropped.

Fixate Information on Click Meta information of a file or commit node is only displayed when the node is hovered. Participants expected that the information of a node gets fixated on click in the stats panel.

Visibility Option Buttons Participants found the labeling of the buttons for fading files or commits confusing. One participant noted that the number of buttons could have been reduced if a toggle switch instead of two buttons for Show Commits/Show Files and Fade Commits/Fade Files had been used. Furthermore, the buttons are clickable when the visualization does not display any data that interacts with the button and therefore has no effect on displayed elements. Figure 6.26 shows how the buttons are labeled and arranged.

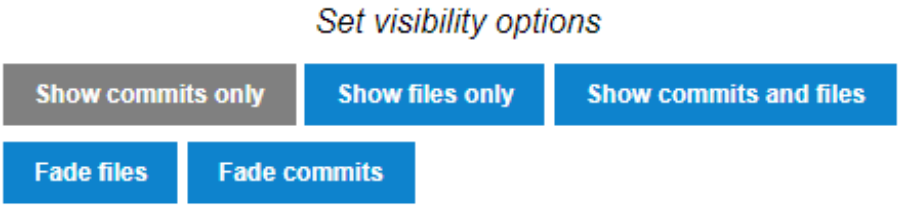


Figure 6.26: Participants were confused by the amount and labeling of buttons for setting the visibility options within the visualization

Behavior of Commit Node Click Clicking on a commit node always visualizes all files that were modified, added or deleted in a commit. Clicking on another commit does not reset previous files but instead adds the files of the newly clicked commit to the visualization. Participants noted that this is a counter-intuitive behavior as they were not sure to which commit the visualized files belong.

Colorization Participants were confused by the color of a file trend representing the file name, compared to the color of a quality metric. However, participants were able to figure out its meaning by reading the legend below the trend chart area. Participants also noted that red should not be used for the visualization of trends as it is a signal color and may indicate problems to novice users.

Propagation between File List and Visualization One participant noted that it would be beneficial to emphasize file trends in the visualization if the mouse cursor hovers over the file in the file list.

Legend Colors and Shapes One participant noted that the description of node shapes, that describes the difference between commit and file nodes should be separated from the color coding, as humans always perceive color before shapes. The participant noted that the

perception of these shapes should be enforced, as they represent a main feature of the visualization.

6.11.7 Improvements

Participants also gave feedback on how to improve the visualization.

Date and Time on Hover Participants suggested displaying the exact date and time of the currently hovered mouse position. It should be displayed in a prominent position like the stats panel.

Quality Metric Threshold To easily tackle out-of-bound commits and files, a participant suggested a configurable interface where a threshold for each metric can be set. All nodes that exceed this threshold are then highlighted with a specific color or shape.

Categorization of File List When clicking on commits, the corresponding files that are listed in the file list should be categorized for each commit. This would make it easier to distinguish which file belongs to which commit.

Relation Trends One participant mentioned that it would be interesting to also see how the relation between quality metrics is trending. He suggested an option to group quality metrics, similar to the current implementation of grouping files to modules.

Undo Functionality Especially participants that clicked on a lot of commit nodes noted that it would be beneficial to have an option to undo the last few actions.

File Version Comparison in Context Menu One participant noted that it is not necessary to specifically select file versions to compare them in the Difference View. It would be easier if the context menu provided two options on right-clicking a file node: compare to the previous version and compare to the next version.

Animated Nodes One of the participants suggested that it would be interesting to see the nodes animating while changing the quality metric. The idea is that every node starts at zero. As soon as a quality metric is switched, the nodes are transitioned to their position on the y-axis. This would give a user a sense of how much a specific node de- or increases when switching between the metrics.

6.11.8 Interpretation of Results

This section interprets the results in regards to the research questions stated in section 6.3.

Research Question 1 The tasks were formulated in a way that every feature from the list in Table 4.1 is covered. All these tasks were fulfilled with a correctness rate of between 80% and 100%, as can be seen in Table 6.12. According to the verbal rating of the SUS, the prototype has a “Good” rating. Therefore, the prototype is useful for inspecting code quality, according to the definition of usefulness in section 3.1.

Research Question 2 Task H specifically aims at answering this research question. It has a correctness rate of 100%. For this given task, users were able to find the exact lines of code that were responsible for the change in quality. The quality metric for this task was *Lines of Code*, therefore users may not be able to find the exact lines of code for other quality metrics without knowing exactly how those metrics are computed.

Research Question 3 All participants have experience in software engineering with an average of 16.5 years. In section 6.11.3 participants rated the practical relevance of each task on a scale from 1 to 5. The average rating from all tasks is 3.97. This emphasises the practical relevance of the prototype.

6.11.9 Threats to Validity

This section discusses possible threats to validity within the evaluation of RepoFlow's visualization.

Quick Start Guide Each participant was instructed to read the quick start guide before executing the tasks. This acts as bias, as participants already knew what to expect before executing each task. Also, participants were allowed to use the quick start guide during the execution of tasks.

Number and Selection of Participants Quantitative analysis of a subject usually requires a large number of randomly selected samples to be able to generalize the results [72]. As already mentioned, the evaluation of this master's thesis only used ten participants.

Wording of Tasks Task J specifically asks for the commit's value of Lines of Code and Comment Lines. However, three participants clicked on the commit and displayed its corresponding file trends. Only one file and therefore only one trend belonged to this commit. Therefore, the value of the commit as well as the file is the same. The three participants read the file's quality value instead of the commit's value, however, it was the correct answer according to the successful completion criteria of the task.

Employed Quality Metrics It was not within the scope of this master's thesis to compute quality metrics but instead visualize them. Therefore, the evaluation was conducted only with five quality metrics: *Cyclomatic Complexity*, *Parameters*, *Statements*, *Lines of Code* and *Comment Lines*.

7 Conclusion

This master's thesis proposed a software visualization that aims at analyzing software quality metric trends and identifying possible causes of change. The idea was based on the specific information needs of software developers that are hard to satisfy. The scope of this thesis was to investigate if the visualization of fine-grained metric trends combined with code difference views satisfies practical information needs in software engineering.

Literature research has revealed that software visualization is an established field with many existing solutions. However, some software visualization categories need further research and development to cover the practical information needs of software engineers. Enabling stakeholders to understand why a project or file has developed in a specific way is one of these categories. Another one is to find out the intention behind a specific code change. Judging if refactoring of existing code is necessary is another highly in demand category.

The problem with current software visualizations is that they are often limited in either trend analysis, incorporation of quality metrics, aggregation of quality metrics, interactivity or the use of finer-grained structures like files or source code. Combining these properties in a new software visualization leads to insights and answers as to how a project has developed during its life-cycle that are hard to answer with current solutions.

The proposed solution in this thesis was a prototype that was, based on the visualization of software quality trends of both commits and files found in a software repository. Commit nodes aggregated quality metrics from all files belonging to a specific commit. File nodes were finer-grained structures of commits and allowed further analysis of commits. As trends incorporated historical source code data, file versions could be compared to each other in an easy way by selecting two different file nodes from the visualization and displaying a source code difference view.

The prototype was evaluated with a scenario-based expert evaluation. Research questions described what was expected to learn from the expert evaluation. After the introduction of the evaluation environment with a demographic questionnaire, a task list, a task rating as well as a SUS questionnaire and an open interview, the results were presented.

Regarding the results of the expert evaluation, most participants stated that they clearly saw an added value of the prototype to existing tools. The general consensus was that the prototype's visualization was more dynamic than current solutions, as it allowed to drill down to a problem cause very fast. Especially participants that used respective existing visualizations said that the prototype would make it easier for them to find relevant information. Summarizing the results from the evaluation, it was shown that the visulization was useful for inspecting code quality metrics and that it enabled users to comprehend why a change of a quality metric happened. The practical relevance of the prototype was emphasized with the results of a task rating. This showed that the visualization of fine-grained metric trends combined with code difference views satisfied practical information needs in software engineering.

7.1 Future Work

The following is a list of features and improvements for the developed prototype. Some of these are direct feedback from the participants of the expert evaluation.

Analyze each line of code for code quality differences Currently, RepoFlow enables the user to view the source code difference between two file versions. If a quality metric changes, the user can compare two file versions that are of interest and see all lines of code that have changed. However, it would be good to not only see all modified lines of code but to see exactly which lines of code have influenced the quality metric. Therefore, a system that can compute and assess, or at least estimate the influence of a single line of code on a quality metric could be beneficial to RepoFlow.

Entity Trends In its current iteration, RepoFlow can analyze files and store this data with its back-end implementation. This limits the quality metrics that can be visualized as that only quality metrics that can be computed on a per-file basis may be visualized. However, the visualization is built in a way that it can handle any artifact in the software engineering process, for example, a class or a function. Generalizing files to entities would greatly enhance the range of use cases that RepoFlow could be utilized for.

Relation Trends Relation trends are a group of two quality metrics that are set in relation to each other. But they are not visualized in the current way of displaying two trends, instead, just one trend represents the related value. These relation trends would be similar to the current feature of files that can be grouped to modules and consequently visualized in the trend chart area.

Configurable Thresholds In software projects it is often important to monitor quality metrics so that they do not exceed certain thresholds. RepoFlow's visualization could incorporate an additional User Interface that allows configuring thresholds for specific metrics. All trends that exceed this threshold in the visualization would be highlighted. This would be another way for users to analyze time periods within a software project that need the focussed attention of stakeholders.

Animated Nodes One of the participants of the expert evaluation suggested that it would be interesting to see the nodes animating while changing the quality metric. The idea is that every node starts at zero. As soon as a quality metric is switched, the nodes are transitioned to their position on the y-axis. This would give a user a sense of how much a specific node decreases or increases when switching between the metrics.

Bibliography

References

[1] Emad Aghajani et al. „The Code Time Machine“. In: *Proceedings of the 25th International Conference on Program Comprehension*. ICPC '17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 356–359. ISBN: 978-1-5386-0535-6. DOI: 10.1109/ICPC.2017.6. URL: https://doi.org/10.1109/ICPC.2017.6.

[2] Selim Aksoy and Robert M Haralick. „Feature normalization and likelihood-based similarity measures for image retrieval“. In: *Pattern recognition letters* 22.5 (2001), p. 565.

[3] Saba Alimadadi, Ali Mesbah, and Karthik Pattabiraman. „Understanding asynchronous interactions in full-stack JavaScript“. In: *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE. 2016, pp. 1169–1180.

[4] Keith Andrews and Johannes Feiner. „RepoVis: Visual Overviews and Full-Text Search in Software Repositories“. In: *2018 IEEE Working Conference on Software Visualization*. 2018.

[5] IEEE Standards Association. *IEEE 1061-1998 IEEE Standard for a Software Quality Metrics Methodology*. IEEE Standards Association, 1998.

[6] Aaron Bangor, Philip Kortum, and James Miller. „Determining what individual SUS scores mean: Adding an adjective rating scale“. In: *Journal of usability studies* 4.3 (2009), pp. 114–123.

[7] Andrew Begel and Thomas Zimmermann. „Analyze this! 145 questions for data scientists in software engineering“. In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 12–23.

[8] Joshua Bloch. „How to design a good API and why it matters“. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM. 2006, pp. 506–507.

[10] John Brooke et al. „SUS-A quick and dirty usability scale“. In: *Usability evaluation in industry* 189.194 (1996), p. 4.

[11] Nanette Brown et al. „Managing technical debt in software-reliant systems“. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM. 2010, pp. 47–52.

[12] David Budgen and Pearl Brereton. „Performing systematic literature reviews in software engineering“. In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 1051–1052.

[13] Raymond PL Buse and Thomas Zimmermann. „Information needs for software development analytics“. In: *Proceedings of the 34th international conference on software engineering*. IEEE Press. 2012, pp. 987–996.

[15] Ward Cunningham. „The WyCash portfolio management system“. In: *ACM SIGPLAN OOPS Messenger* 4.2 (1993), pp. 29–30.

[16] Tommaso Dal Sasso et al. „Blended, not stirred: Multi-concern visualization of large software systems“. In: *Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference on*. IEEE. 2015, pp. 106–115.

[17] Karen Daniels et al. „Properties of normalized radial visualizations“. In: *Information Visualization* 11.4 (2012), pp. 273–300.

[18] Michael C Fanning and Nicholas Guerrera. *Iterative static and dynamic software analysis*. US Patent 7,975,257. 2011.

[19] Laura Faulkner. „Beyond the five-user assumption: Benefits of increased sample sizes in usability testing“. In: *Behavior Research Methods, Instruments, & Computers* 35.3 (2003), pp. 379–383.

[20] Norman E Fenton and Martin Neil. „Software metrics: successes, failures and new directions“. In: *Journal of Systems and Software* 47.2-3 (1999), pp. 149–157.

[22] Michael Friendly and Daniel J Denis. „Milestones in the history of thematic cartography, statistical graphics, and data visualization“. In: *URL http://www. datavis. ca/milestones* 32 (2001), p. 13.

[23] Daniel Galin. *Software quality assurance: from theory to implementation*. Pearson Education India, 2004.

[24] Martin Glinz and Roel J. Wieringa. „Stakeholders in Requirements Engineering“. In: *IEEE Software* 24 (2007), pp. 18–20.

[25] Wolfgang Globke. „Software-Metriken“. In: *Moderne Softwareentwicklung, Universität Karlsruhe* (2005), p. 8.

[27] Jeffrey O. Grady. „2 - Requirements Foundation“. In: *System Requirements Analysis (Second Edition)*. Ed. by Jeffrey O. Grady. Second Edition. Oxford: Elsevier, 2014, pp. 93 –150. ISBN: 978-0-12-417107-7.

[28] Lars Grammel, Melanie Tory, and Margaret-Anne Storey. „How information visualization novices construct visualizations“. In: *IEEE transactions on visualization and computer graphics* 16.6 (2010), pp. 943–952.

[29] Jonathan Grudin. „Utility and usability: research issues and development contexts“. In: *Interacting with computers* 4.2 (1992), pp. 209–217.

[30] Robert L Harris. *Information graphics: A comprehensive illustrated reference*. Oxford University Press, 2000.

[31] Lane Harrison et al. „Ranking Visualizations of Correlation Using Weber's Law.“ In: *IEEE Trans. Vis. Comput. Graph.* 20.12 (2014), pp. 1943–1952.

[32] Jeffrey Heer, Michael Bostock, Vadim Ogievetsky, et al. „A tour through the visualization zoo.“ In: *Commun. Acm* 53.6 (2010), pp. 59–67.

[34] Stacy A Jacob and S Paige Furgerson. „Writing interview protocols and conducting interviews: Tips for students new to the field of qualitative research“. In: *The qualitative report* 17.42 (2012), pp. 1–10.

[35] Mohd Ehmer Khan et al. „Different approaches to white box testing technique for finding errors“. In: *International Journal of Software Engineering and Its Applications* 5.3 (2011), pp. 1–14.

[36] Muzammil Khan and Sarwar Shah Khan. „Data and information visualization methods, and interactive mechanisms: A survey“. In: *International Journal of Computer Applications* 34.1 (2011), pp. 1–14.

[37] Barbara Kitchenham. „Procedures for performing systematic reviews“. In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26.

[39] Andrew J Ko, Robert DeLine, and Gina Venolia. „Information needs in collocated software development teams“. In: *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society. 2007, pp. 344–353.

[40] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. „Exploring the evolution of software quality with animated visualization“. In: *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*. IEEE. 2008, pp. 13–20.

[41] Thomas D LaToza and Brad A Myers. „Hard-to-answer questions about code“. In: *Evaluation and Usability of Programming Languages and Tools*. ACM. 2010, p. 8.

[43] Chang Liu, Xin Ye, and En Ye. „Source Code Revision History Visualization Tools: Do They Work and What Would It Take to Put Them to Work?“ In: *IEEE Access* 2 (2014), pp. 404–426.

[45] Robert Martin. „OO design quality metrics“. In: *An analysis of dependencies* 12 (1994), pp. 151–170.

[46] Leonel Merino, Mohammad Ghafari, and Oscar Nierstrasz. „Towards actionable visualisation in software development“. In: *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*. IEEE. 2016, pp. 61–70.

[48] Karine Mordal et al. „Software quality metrics aggregation in industry“. In: *Journal of Software: Evolution and Process* 25.10 (2013), pp. 1117–1135.

[49] Haris Mumtaz, Beck Fabian, and Daniel Weiskopf. „Detecting Bad Smells in Software Systems with Linked Multivariate Visualizations“. In: *2018 IEEE Working Conference on Software Visualization*. 2018.

[50] Emerson Murphy-Hill and Andrew P Black. „An interactive ambient visualization for code smells“. In: *Proceedings of the 5th international symposium on Software visualization*. ACM. 2010, pp. 5–14.

[51] Brad A Myers and Jeffrey Stylos. „Improving API usability“. In: *Communications of the ACM* 59.6 (2016), pp. 62–69.

[52] Nachiappan Nagappan and Thomas Ball. „Use of relative code churn measures to predict system defect density“. In: *Proceedings of the 27th international conference on Software engineering*. ACM. 2005, pp. 284–292.

[53] Jakob Nielsen. *Usability engineering*. eng. Interactive Technologies. Cambridge, Mass.: AP Professional, 1993. ISBN: 0080520294.

[54] Martin Pinzger et al. „Visualizing multiple evolution metrics“. In: *Proceedings of the 2005 ACM symposium on Software visualization*. ACM. 2005, pp. 67–75.

[55] Rolf Porst. *Im Vorfeld der Befragung: Planung, Fragebogenentwicklung, Pretesting*. Vol. 1998/02. 1998, p. 44.

[56] Patrick Riehmann, Manfred Hanfler, and Bernd Froehlich. „Interactive sankey diagrams“. In: *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE. 2005, pp. 233–240.

[57] George Robertson et al. „Effectiveness of animation in trend visualization“. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008).

[58] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: how to plan, design and conduct effective tests*. John Wiley & Sons, 2008.

[59] Thomas P Ryan and JP Morgan. „Modern experimental design“. In: *Journal of Statistical Theory and Practice* 1.3-4 (2007), pp. 71–80.

[61] Francisco Servant and James A Jones. „Chronos: Visualizing slices of source-code history“. In: *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*. IEEE. 2013, pp. 1–4.

[62] Ben Shneiderman. „The eyes have it: A task by data type taxonomy for information visualizations“. In: *The Craft of Information Visualization*. Elsevier, 2003, pp. 364–371.

[64] Diomidis Spinellis. „Version Control Systems“. English. In: *IEEE Software* 22.5 (2005). Copyright - Copyright Institute of Electrical and Electronics Engineers, Inc. (IEEE) Sep/Oct 2005; Dokumentbestandteil - illustrations; Zuletzt aktualisiert - 2014-05-18; CODEN - IESOEG, pp. 108–109.

[65] Maurice Termeer et al. „Visual exploration of combined architectural and metric information“. In: *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on*. IEEE. 2005, pp. 1–6.

[66] Eva Van Emden and Leon Moonen. „Java quality assurance by detecting code smells“. In: *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*. IEEE. 2002, pp. 97–106.

[67] Erik Van Zijst et al. *Merge previewing in a version control system*. US Patent 9,430,229. 2016.

[69] Moritz Wittenhagen, Christian Cherek, and Jan Borchers. „Chronicler: Interactive exploration of source code history“. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM. 2016, pp. 3522–3532.

[70] Claes Wohlin. „Guidelines for snowballing in systematic literature studies and a replication in software engineering“. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. ACM. 2014, p. 38.

[71] Ji Soo Yi, Youn ah Kang, and John Stasko. „Toward a deeper understanding of the role of interaction in information visualization“. In: *IEEE transactions on visualization and computer graphics* 13.6 (2007), pp. 1224–1231.

[72] Kaya Yilmaz. „Comparison of quantitative and qualitative research traditions: Epistemological, theoretical, and methodological differences“. In: *European Journal of Education* 48.2 (2013), p. 313.

Online References

[9] Stefan Brass. *Objektorientierte Programmierung*. 2013. URL: http://users.informatik.uni-halle.de/~brass/oop13/p8_state.pdf (visited on 04/20/2019).

[14] Michaël Van Canneyt. *Free Pascal*. 2015. URL: https://www.freepascal.org/docs-html/3.0.0/ref/refse2.html (visited on 04/20/2019).

[21] Apache Software Foundation. *A version control glossary*. URL: https://www.openoffice.org/docs/ddCVS_cvsglossary.html.ko (visited on 04/19/2019).

[26] Google. *Issues*. 2017. URL: https://developers.google.com/issue-tracker/concepts/issues (visited on 04/19/2019).

[33] Seerene Inc. *Seerene*. 2019. URL: https://seerene.com/ (visited on 04/20/2019).

[38] Kiuwan. *Kiuwan*. 2019. URL: https://www.kiuwan.com/ (visited on 04/20/2019).

[42] Inc. Linux Kernel Organization. *Git User Manual*. URL: https://mirrors.edge.kernel.org/pub/software/scm/git/docs/user-manual.html (visited on 07/26/2018).

[44] Jean-François Lépine. *PHPMetrics*. 2015. URL: https://www.phpmetrics.org/ (visited on 04/20/2019).

[47] Microsoft. *Differences Between Parameters and Arguments*. 2015. URL: https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/procedures/differences-between-parameters-and-arguments (visited on 04/20/2019).

[60] Seerene. *Seerene KPI Definitions*. URL: https://www.seerene.com/wp-content/uploads/2017/09/Seerene-KPI-Definitions-Sept-2017.pdf (visited on 07/26/2018).

[63] SonarSource. *SonarQube*. URL: https://www.sonarqube.org/ (visited on 04/20/2019).

[68] David A. Wheeler. *More Than a Gigabuck: Estimating GNU/Linux's Size*. 2002. URL: https://dwheeler.com/sloc/redhat71-v1/redhat71sloc.html (visited on 04/20/2019).

A Appendix

	0	1	2	3	4	5
Exemplary practical use case for Task A: analyze quality metrics of specific commits.	☐	☐	☐	☐	☐	☐
Exemplary practical use case for Task B: analyze quality metrics of specific files for specific points in time.	☐	☐	☐	☐	☐	☐
Exemplary practical use case for Task C: search for files with highest/lowest quality values within a commit.	☐	☐	☐	☐	☐	☐
Exemplary practical use case for Task D: analyze the impact of single files on the overall quality value of a commit.	☐	☐	☐	☐	☐	☐
Exemplary practical use case for Task E: Compare multiple quality trends of the same file and relate the development of different software quality metrics.	☐	☐	☐	☐	☐	☐
Exemplary use case for Task F: analyze the quality value of a module consisting of multiple files.	☐	☐	☐	☐	☐	☐
Exemplary use case for Task G: read out meta information of specific file versions.	☐	☐	☐	☐	☐	☐
Exemplary use case for Task H: find the source code change that is responsible for a change in a quality metrics trend.	☐	☐	☐	☐	☐	☐
Exemplary practical use case for Task I: analyze densely clustered files in the visualization (via zooming in on the time axis) and read out their quality values.	☐	☐	☐	☐	☐	☐
Exemplary practical use case for Task J: compare different quality metrics for specific commits.	☐	☐	☐	☐	☐	☐

Table A.1: Task rating questionnaire to assess the practical value of each task

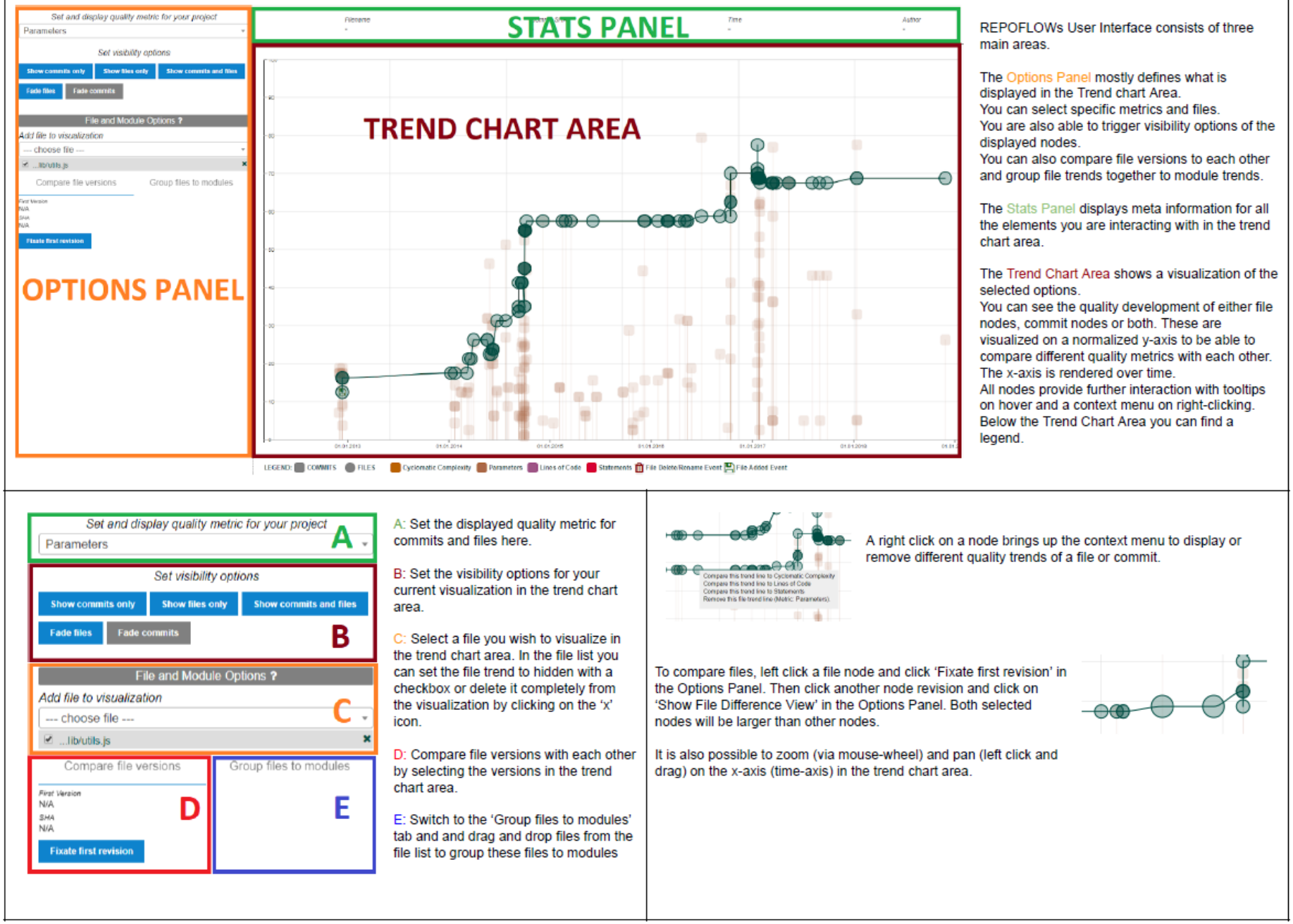


Figure A.1: The Quick Start Guide for REPOFLOW. Each participant gets a copy before executing the tasks from the task list.

I think that I would like to use this website frequently.	☐	☐	☐	☐	☐
I found this website unnecessarily complex.	☐	☐	☐	☐	☐
I thought this website was easy to use.	☐	☐	☐	☐	☐
I think that I would need assistance to be able to use this website.	☐	☐	☐	☐	☐
I found the various functions in this website were well integrated.	☐	☐	☐	☐	☐
I thought there was too much inconsistency in this website.	☐	☐	☐	☐	☐
I would imagine that most people would learn to use this website very quickly.	☐	☐	☐	☐	☐
I found this website very cumbersome/awkward to use.	☐	☐	☐	☐	☐
I felt very confident using this website.	☐	☐	☐	☐	☐
I needed to learn a lot of things before I could get going with this website.	☐	☐	☐	☐	☐

Table A.2: SUS questionnaire - the range of the checkboxes is 'strongly disagree' in the outer left to 'strongly agree' in the outer right checkbox

What is your experience with software engineering in years?						
	0	1	2	3	4	5
How would you rate your experience with software repositories on a scale from 0 (zero) to 5 (five) with 5 being very experienced and 0 not experienced?	☐	☐	☐	☐	☐	☐
How would you rate your experience with software quality metrics on a scale from 0 (zero) to 5 (five) with 5 being very experienced and 0 not experienced?	☐	☐	☐	☐	☐	☐
Gender	☐ male		☐ female		☐ other	
Age						
I consent to screen/audio recordings being made of this testing session for evaluation purposes.	☐ Yes			☐ No		

Table A.3: Demographic Questionnaire for the expert evaluation