Universität Zürich[UZH]

Department of Informatics

Bachelors's Thesis

Extension of ReviewVis: Adaption of the Code-Review Visualization Tool for Industry Partner Mozilla

Marc Kramer
15-917-941
marc.kramer@uzh.ch

supervised by
Prof. Dr. Alberto Bacchelli
Enrico Fregnan
Zurich Empirical Software Engineering Team

31st July 2022

Abstract

Code review is a widespread practice used to improve code quality and maintainability, enable knowledge transfer between co-workers, and generally reduce defects in software created when writing and modifying it. By giving the engineers adequate tools, their achieved effectiveness in reviewing code can be improved. This is why we extended the tool ReviewVis with the goal of making it possible for Mozilla to use it. During our work, we also changed its architecture to make it easily extendable in the future. In addition, we evaluated the acceptance of the visualization technique used in ReviewVis. We set it against other existing approaches to visualizing code changes by conducting semi-structured interviews. The participants responded well to ReviewVis and preferred it over the others, even in situations where it reaches its limitations.

Zusammenfassung

Code Review ist eine heutzutage weitverbreitete Methode, um die Qualität von Sourcecode und dessen Wartungsfreundlichkeit zu verbessern, sowie den Wissensaustausch zwischen Mitarbeitenden zu ermöglichen und generell die Anzahl Fehler in der Software zu reduzieren. Durch das Bereitstellen von Hilfsmitteln, welche die Entwickler während des Reviews nutzen können, kann deren Effizient weiter gesteigert werden. Darum haben wir das Tool ReviewVis, mit dem Ziel es für Mozilla einsetzbar zu machen, erweitert. Während unserer Arbeit, haben wir die Architektur von ReviewVis so angepasst, dass es in Zukunft leicht erweitert werden kann. Zusätzlich dazu, haben wir mittels Interviews die allgemeine Akzeptanz der von ReviewVis verwendeten Visualisierungstechnik gegenüber deren anderer Tools bei Entwicklern evaluiert. Die Teilnehmer reagierten gegenüber ReviewVis positiv und bevorzugen dessen Visualisierung auch in Situationen, in denen es seine Grenzen erreicht.

Contents

List of Tables

List of Figures

1 Introduction

Whether in a professional or an open-source environment, the practice of code review is a ubiquitous, longstanding process carried out by many software engineers. The goals of reviewing changed code before integrating it into an existing code base include reducing possible errors and flaws it contains, making it more maintainable and future-proof, and improving the overall code quality [Bacchelli and Bird, 2013]. Further, and not to be undervalued, benefits are knowledge transfer and shared ownership of the codebase. These two benefits are automatically established between the participating engineers during the process of reviewing code changes.

For some developers, doing code reviews represents a fundamental element next to their other daily activities. Optimizing the processes during these reviews and thereby increasing their efficiency can have a significant impact on how developers spend their time during software engineering. However, today's lightweight and asynchronous approach used during modern code review has not always been the standard procedure. In the early days of reviewing code, it was conducted in a more rigid and strict manner [Fagan, 1976]. This rigorous process was relaxed in recent years, and the prior mentioned style of doing modern code review became the new norm.

Multiple aspects of code review have already been thoroughly investigated to increase their effectiveness and developers' efficiency during the execution of it. Rigby et al. studied how the number of reviewers influences efficiency and effectiveness and found that having too many reviewers can lead to negative impacts [Rigby et al., 2014]. The study done by dos Santos and Nunes, validated this [dos Santos and Nunes, 2017]. In addition to the influence of the number of reviewers, dos Santos and Nunes discovered that the number of changes and the number of teams affected can also greatly influence the duration of modern code reviews. On the contrary, there are proven methods to facilitate and improve the review process. The development and deployment of tools that automate tasks of the reviewer, like static code analysis, are a great example of this [Balachandran, 2013]. Such tools often not only reduce the workload but also work as a safety net that could catch possible oversights of the developer and the reviewer. But these tools can not only bring beneficial aspects with them. Söderberg et al. took an in-depth look at how the misalignment of support tools used during code review can even lower the effectiveness and efficiency when not used correctly [Söderberg et al., 2022]. They discovered that in some cases, the tools could bring major obstacles with them, e.g., not providing the right information needed, not doing the automated work the reviewer expects from them, and the tools not being suited for the size of the code review.

In this paper, we present our extension to ReviewVis, which was done concerning the interest of Mozilla [1] adopting the tool in their review procedure. In the same take, we wanted to address possible friction areas found by Söderberg et al. in ReviewVis, and we wanted to modify it to make it more expandable in the future. In a second phase, we took a step back. We intended to find out if the visual approach ReviewVis takes to support developers is well-received by them and if they would use the tool. To lay the foundation for the analysis, we evaluated multiple other change visualization tools that share the intention with ReviewVis to support engineers during code review. We then compared these tools to ReviewVis. To gain further insights on the benefits and also possible flaws, we conducted a study where we interviewed a total of six developers. We hoped to gather insight into their overall preferences when using such tools and if ReviewVis would appeal to them more than the other tools. Two-thirds of the study participants favored ReviewVis over the other tools we presented to them. They also chose ReviewVis over the other tools in some situations where it was known for it to reach its limitations.

[1] `https://www.mozilla.org/`

2 Background and Related Work

The first part of this section aims to give the reader a brief overview of different types of existing tools that aim to improve effectiveness and efficiency when reviewing code. In Section 2.2, we specifically present tools that rely on visualization to achieve this. One of these visualization-based tools that intend to support developers during code review is ReviewVis. Since we extended it during our work on this thesis, it is introduced in its own section.

2.1 Existing Code Review Support Tools

As presented in the introduction, the range of available support tools is extensive. However, the actual phase of a review where they can be used in can differ. Some tools make the review process more effective by performing static analysis of the code in advance of the manual review [Singh et al., 2017], and other tools, like GitLab[2], have the ability to run tasks after a review is completed. This thesis will focus on tools that actively support engineers during practicing code review. Baum et al. found out that tools that group files containing related changes, rather than listing them alphabetically, increase efficiency by reducing the cognitive load of the developers [Baum et al., 2017]. Dias et al. developed a tool that untangles merge requests, i.e., splits them based on how they change the code [Dias et al., 2015]. With it, refactoring changes and changes for fixing bugs are split, making them easier to understand. The efficiency of code reviews can also be increased by maximizing the number of defects that are found during it. A possible way of achieving that is by using a checklist that the developer can follow [Cohen et al., 2006]. To be able to analyze the way ReviewVis enables developers to be more efficient, we will introduce a selection of similar tools which also use visualization to achieve this.

2.2 Visualization-Based Support Tools

Before presenting the tool we extended, we provide an overview of related tools that can help the reviewer during the reviewing process by providing additional information visually. Similar to the general tools mentioned before, the list of possible selections for graphical tools is long, and the method they utilize to display the additional information can significantly vary. To cover as much area as possible, we intentionally chose tools that use diverse visualization techniques.

2.2.1 ChangeVis

The method applied by ChangeVis [Gasparini et al., 2021] enhances the differential viewer of source control management (SCM) systems. The tool can, for example, be linked to GitHub[3], where it adds two columns next to blocks of code in the *Changed Files* tab. The column on the left contains links to references of objects contained within the block of code where the change was made in. The right column shows links to definitions contained in that block. An example of the two added columns can be seen below in Figure 1. By clicking on these links, the tool opens the file containing the linked reference or definition in a separate window. The authors created the tool in an iterative fashion and conducted a series of interviews and a study between the iterations. They argue the tool reduces two of the most common burdens based on [Bacchelli and Bird, 2013] during code review, which are understanding the changes in a change set, and understanding the impact of the changes.

[2] `https://about.gitlab.com/`
[3] `https://github.com/`

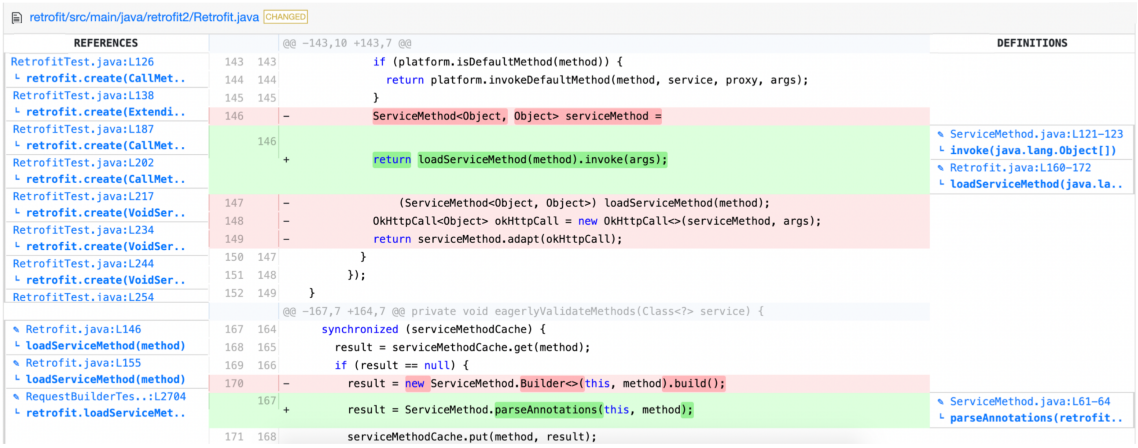


Figure 1: Interface of ChangeVis [Gasparini et al., 2021]

2.2.2 ViDI

Another example of a tool that uses visualization as its primary means is called Visual Design Inspection, in short ViDI [Tymchuk et al., 2015]. Compared to ChangeVis, the way ViDI displays the additional information is significantly different. It makes use of the so-called city-based visualization technique [Wettel et al., 2011]. The classes are laid out in a block-based grid, where they serve as foundations. The methods contained in the classes are stacked upon each other above them, resulting in a visualization resembling a real-life city with skyscrapers. The components of the graph are also interactive, and the reviewer can hover or click on different elements to see further information about them or drag the graph around to look at it from different angles. Additionally, the foundations and the cubes are also color-coded, which presents the reviewer with immediate information about an attribute of them. An image of the interface of ViDI can be found in Figure 2.

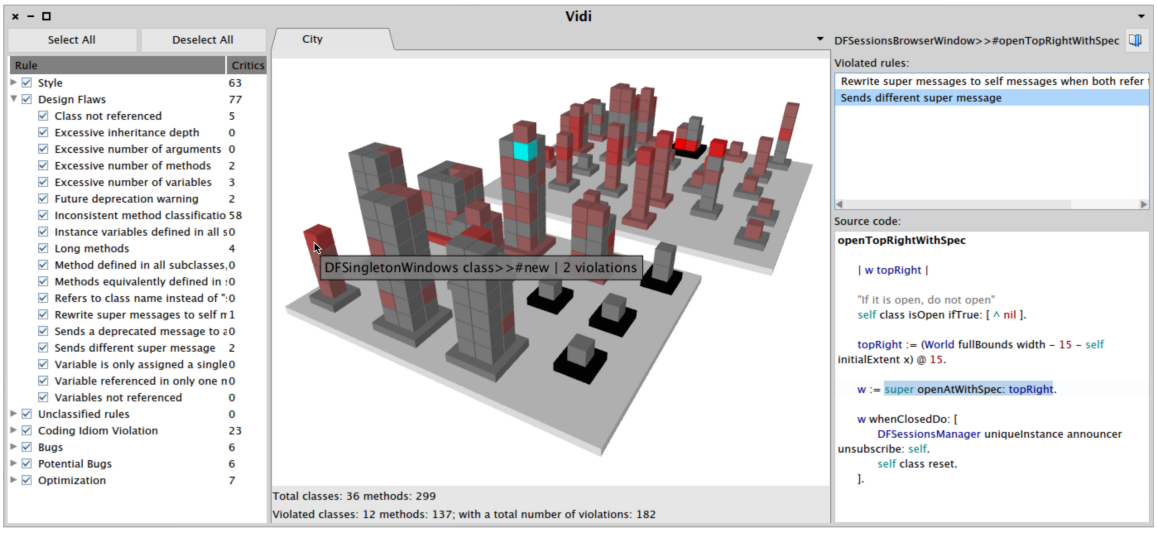


Figure 2: Interface of ViDI [Tymchuk et al., 2015]

2.2.3 Softagram

The following visualization-based tool we present is called Softagram, which was created by the same-named company Softagram[4]. Similar to ChangeVis, it can also be added to various SCM platforms. Compared to the other tools, it, however, differs distinctively in how it presents the visualization. Instead of directly showing the graph next to the code changes, it analyses the code first and then adds the results, including different charts, as a comment to the pull request. An example of such a comment can be found in Figure 3. To gain further insights, the developer can then open the graphs in a desktop app. Other than that, the graph does not provide more possibilities to interact with it.

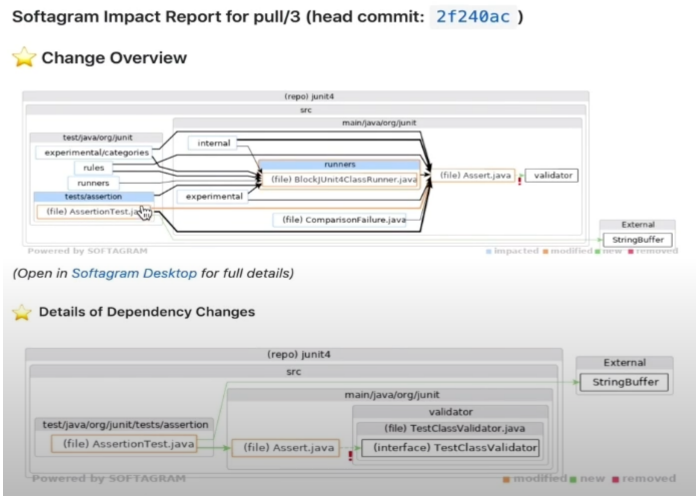


Figure 3: Generated comment by Softagram [4]

The next graph-based tool we present is called ReviewVis. Since it is central to this thesis, we introduce it in its own section.

2.3 ReviewVis

The tool we extended during this thesis is called ReviewVis and was initially created by Fröhlich as a part of his master thesis [Fröhlich, 2020]. The goal was to analyze if and how using a call and dependency graph as a supportive figure would benefit developers during their review. To answer these questions, he created a tool called ReviewVis. The result of multiple iterations of testing phases and surveys was a tool that shows highly interactive graphs in a separate window next to a merge request opened in GitLab.

An example of such a graph can be found in Figure 4 below. The circles represent changed classes, and their names are placed in a rectangle in their center. Next to them are boxes with rounded corners that represent methods contained within these classes, connected to each other by dotted lines. The fully stroked lines represent dependencies between the classes. The color coding can be interpreted as green means added, orange means changed, and red means removed. Further, the generated classes and methods are colored in gray, and files with changes that could not have been parsed are colored in blue.

[4] https://softagram.com/

When conducting a review while using ReviewVis, the developer also has the ability to directly modify the graph by interacting with it. Next to moving the different nodes around by dragging them, they can remove them by a right-click, hover over them, or the belonging change in the GitLab window to highlight the node and its connections.

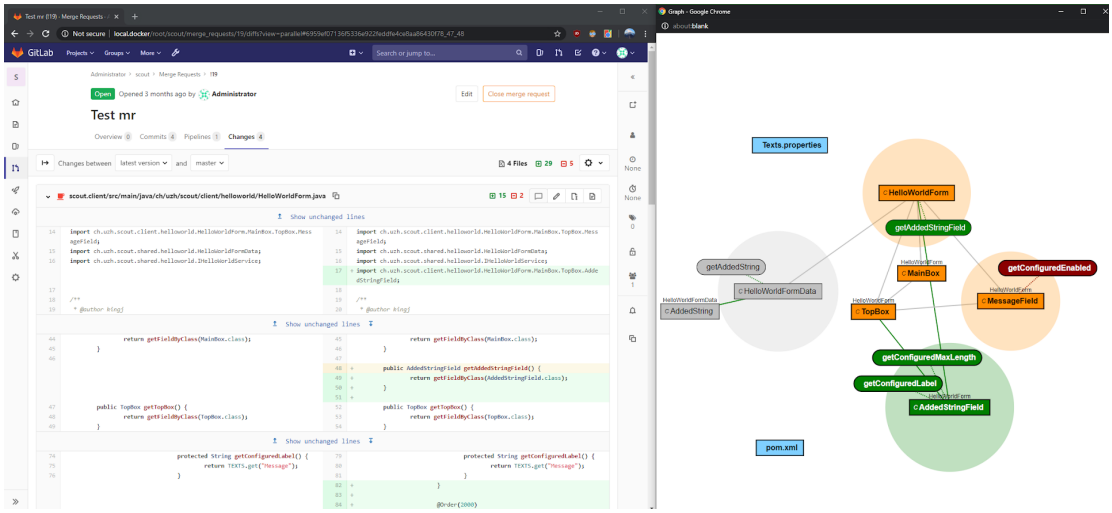


Figure 4: ReviewVis next to an open merge request [Fröhlich, 2020]

The internal structure of the tool consists of two main parts, the CodeDiffParser (CDP) and the CodeDiffVisualizer (CDV). As the name suggests, the CDP is responsible for parsing the changed source code. It uses abstract syntax trees (AST) to compare the source and target branches of merge requests with each other. The thereby found differences are then structured and written to a JSON file. The CDV then uses this JSON file to display the graph. While the CDP is originally invoked by entering command line commands, CDV is installed as a Google Chrome[5] extension. The popup of the extension is shown in Figure 5, where we can also see the available settings to customize the graph.

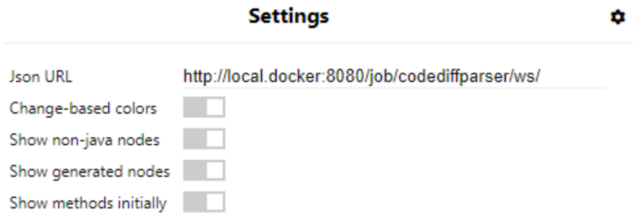


Figure 5: Popup of CDV 1.0 created [Fröhlich, 2020]

The tool was already once modified by Botschen during his bachelor thesis [Botschen, 2022]. He extended the CDP by adding a Python[6] parser. In the same take, the CDV was also improved, such that it is possible to enable displaying different languages in multiple windows, respectively. The version created by Botschen also served as the basis of our extension, which we present in the next chapter.

[5] https://www.google.com/intl/en_us/chrome/
[6] https://www.python.org/

3 Extension of ReviewVis

In this section, we present the modifications we implemented into ReviewVis and the reasons behind them. In Section 3.1, we first present the requirements we received from Mozilla and set ourselves. In the same section, we present the goals we wanted to achieve based on the requirements and the plan we created to do so. In Section 3.2, we provide an overview of the general changes made to the architecture and internal workings of ReviewVis, including the difficulties we ran into while doing so. In Section 3.3 we summarize the work we did and in Section 3.4 we present future work for ReviewVis.

3.1 Requirements

As mentioned in the introduction of the thesis, the origin of the need to modify ReviewVis came, next to our own interest, from Mozilla. They are interested in integrating the tool into their workflows and discovering how it influences the effectiveness and efficiency of their software engineers when they perform code reviews. For them to be able to use the tool, however, it needed some fundamental changes to its compatibility, architecture, and functionality. To plan our modification process for these needed changes, they provided us with requirements and, in addition, some feature requests they liked to see.

The first and most evident requirement was to modify the CDV Chrome extension to run in Mozilla's browser Firefox[7]. The exact steps we took when porting the extension will be described in Section 3.2. Next, ReviewVis just worked with merge requests opened in GitLab, but since Mozilla uses Phabricator[8] as their SCM platform of choice, we also had to add support for it. We will present the modifications we made to CDV to achieve Phabricator support in Section 3.2 too. The last obligatory change we needed to make was to add support for additional programming languages. For this, Mozilla provided us with the languages they use most. This added C++[9], JavaScript[10], and Rust[11] to the list our tool should support. Since adding all three languages would have been beyond the scope of this thesis, we decided to add their most used one, which is C++. As mentioned before, in addition to the obligatory requirements, Mozilla also stated some nice-to-have features to add. Since we could not include them in our work due to time constraints, we will briefly cover them in the list of future additions in Section 3.4. We still listed them together with the prior mentioned requirements in Table 1 .

Requirement	Origin	Priority
Port extension to Firefox	Mozilla	Necessary
Support for Phabricator	Mozilla	Necessary
Support parsing C++, JavaScript or Rust	Mozilla	Necessary
Add option to filter renaming changes	Mozilla	Facultative
Add option to filter test cases	Mozilla	Facultative
Refactor into extendable architecture	ZEST	Necessary
Reduce manual steps	ZEST	Facultative

Table 1: Overview of requirements

[7] https://www.mozilla.org/en-US/firefox/browsers/
[8] https://www.phacility.com/phabricator/
[9] https://isocpp.org/
[10] https://www.ecma-international.org/publications-and-standards/standards/ecma-262/
[11] https://www.rust-lang.org/

As mentioned in the introduction of this section, we also took the opportunity to add some requirements of our own to the list. Before starting this thesis, Botschen added Python to the list of supported languages of ReviewVis. Since we expected the tool to be extended with further parsing support for additional languages, we wanted to change its parsing architecture fundamentally to make such additions a straightforward process. Next, we also wanted to eliminate as much manual work as possible for the developers when they use the tool. We hoped to increase their efficiency even further with this. After presenting the requirements, we will now take an in-depth look at the process of implementing them.

3.2 Modification Process

This subsection presents the steps we took to implement the requirements listed above. Before looking at the individual steps, we give a high-level overview of the modified architecture we planned to implement.

3.2.1 Architecture

To reduce the manual steps for the user between entering the required inputs and displaying the thereof-based graph, we modified CDP to receive the necessary information for parsing directly from CDV. An illustration of the high-level overview of the planned process can be found below in Figure 6. In our planned architecture, the user enters the needed values directly into CDV via the extension (1) instead of invoking CDP by executing command line prompts. Next, the CDP will fetch the requested files directly from the Phabricator API (2) by performing a series of API calls. Once the files are retrieved (3), CDP runs them through the parser for their respective language (4) and creates the JSON containing the information about the changes. The CDV then receives the information that the parsing is completed (5) and can display the graph (6).

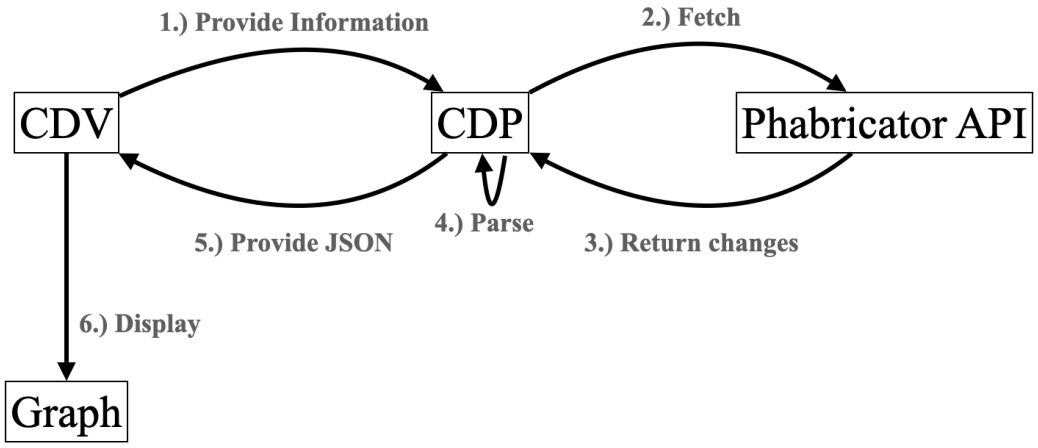


Figure 6: Process overview

These fundamental changes in how we planned the CDP and CDV to interact with each other required us to make extensive modifications to the code base of ReviewVis. Before looking at how we transformed CDP, we start by presenting our initial work on the existing browser extension of CDV.

3.2.2 Firefox Extension

We started our work by modifying the existing Chrome extension to make it launch in Firefox. Since the APIs of both browsers are designed with interoperability between both environments, this initial step could be completed nearly straightforwardly. The only issue we ran into was with the `declarativeContent` call. This call is only available in the Chrome API and does not exist within the API of Firefox. After replacing its logic with available API calls from Firefox, the extension's popup displayed without further problems. Our next goal was to find a way to communicate between the CDV and CDP components directly.

3.2.3 Native Messaging

The solution we chose for exchanging information between the browser extension and our parser is called Native Messaging[12]. Native Messaging allows browser extensions to directly exchange messages with pre-defined applications that are locally installed on the user's computer. We chose to implement this solution to remove the need to invoke the two parts of ReviewVis separately, thereby eliminating manual steps the user needs to make.

Native Messaging works by the extension establishing a connection using the `runtime` API of the browser. As soon as this connection is established, we can send messages from the extension, which are received by the native application on the standard input `stdin.read()`. The native application can then send information back again using the standard output. These messages are then in turn received by the extension through `runtime.port.onMessage()`. Figure 7 presents a graphical overview of the interactions.

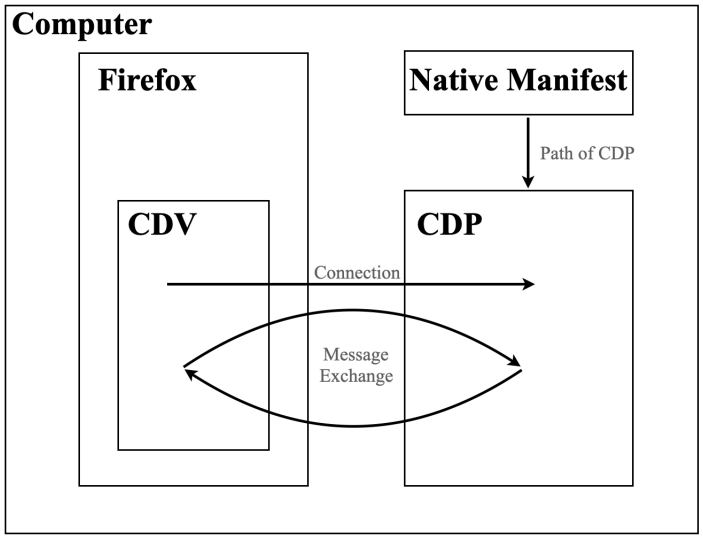


Figure 7: Native Messaging overview, based on Native Messaging Documentation[12]

[12] `https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging`

For the extension to be able to set up such a connection, we first had to create a JSON file called the *Native Manifest*. This manifest must contain specific metadata such as the extension name, its identifier, and the local path of the application. Saving this manifest must be done in the right location and is not done automatically. Also, this location where it must be saved is operating system dependent.

To test sending such messages, we added some provisional text input fields to the extension popup. We planned to use them later for entering the required information for fetching files from Phabricator. After enabling the possibility to send rudimentary messages as JSON from the extension, we continued to modify the CDP to be able to receive them.

3.2.4 Phabricator API

Before modifying the CDP, we made a proof-of-concept backend to try out the native messaging functionalities mentioned before. As a reference, we used the instructions provided by Mozilla on their developer webpage about Native Messaging. After we succeeded in sending messages forth and back from both CDP and CDV, we started to integrate the fetching of the required files for parsing from Phabricator. We chose to create an automatic fetcher to finish the removal of manual steps for the developers. As mentioned, our goal was for users just to enter information into the CDV, and the two components would handle the rest for them. To replace the manual entry of branch info, we tried to find a way we could retrieve the changed files directly from Phabricator only using the differential identifier. As it turned out, working with the Phabricator API was a more complicated task than anticipated. We discovered that there was no direct API call to retrieve all the changed files from a differential. To test the API, there exists the option to try out the API calls directly from the documentation interface called Conduit [13]. However, the error messages received there did not provide accurate information on the root cause of the error. Instead, we found that testing the API via console commands returned more informative error messages. We are not sure if this is related to the fact that Phabricator is no longer being actively maintained since June 1, 2021[14].

In the end, we managed to find a path of making multiple, sequential API calls that allows us to download the target branch's original files together with the differential's raw diff. However, in addition to just providing the differential identifier, the user must also enter the current diff-number and the parent commit identifier. This is undoubtedly suboptimal since it re-adds more manual work, but the needed inputs can all be quickly found on the differential page itself and could most certainly be automated in the future. After fetching the original files and the raw diff, we apply the diff to the files and save them next to their original ones. Adding this functionality completed our work concerning Phabricator's API.

3.2.5 Refactoring CDP

The next task was to modify the existing CDP such that further language support could be easily added. As mentioned in Section 2.3, Botschen added Python and functional language support to the parser during his extension. Since he added the Python parser using a clean implementation, we decided to use it as the foundation on which we would base our work.

[13] `https://phabricator.services.mozilla.com/conduit/`

[14] `https://admin.phacility.com/phame/post/view/11/phacility_is_winding_down_operations/`

We first began by integrating the native messaging and Phabricator API fetcher prototype into the existing project. Instead of launching the parser directly from the command line, it will now be started as soon as the fetcher finished retrieving the files and applied the diff. Next, we analyzed the inner workings of the Python parser to determine for which components it made sense to be factored out in order to make them generally available and usable by different parsers. We also looked at how the existing helper functions could be modified to be used in multiple parsers. But the most crucial point was to find out where we could integrate the logic that handles the distribution of the retrieved files to their respective parser. The Python parser already had a file check to filter out the non-Python files. We refactored this check and moved the parsing of Python files into its own classes. Like this, one can simply add new language types to the checker and invoke the corresponding parser from it. The updated architecture can be found in Figure 8. With this work done, we could now focus on adding the new C++ parser itself.

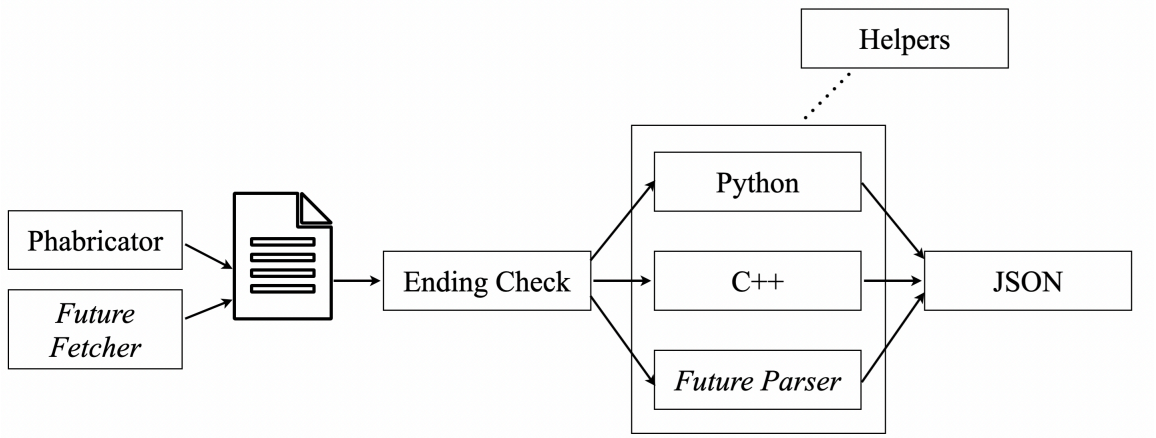


Figure 8: Updated architecture of CDP

3.2.6 Extending CDP

To be able to choose the most suitable parser for C++, we first did some research on existing ones. During this research, we came across a blog post by Bendersky dating back from 2011 [Bendersky, 2011]. In this blog post, she explains the fundamentals of using the `libclang` Python bindings to analyze the AST of the C-based languages C++, Objective-C, and C. As Bendersky stated in her blog post, working with these bindings is not a trivial task. In this post, she also showed the code snippets she created at that time. However, the code in the blog post was severely outdated and did not work anymore, but it still served as valuable guidance for our implementation. The official documentation of the Python bindings is an over 4000 lines of code (LOC) long file containing generated documentation. Finding the correct method or variable to extract the desired info from the AST was tedious.

In the end, the way our implementation works can be described as follows: We use a cursor provided by the `clang` compiler to traverse all entities on the AST. Like the existing parsers in ReviewVis, we filter out the elements of interest, like classes and function definitions. We then create nodes for each of the cursor entities by collecting info by using provided variables or functions of the bindings. Finally, these nodes are then added to the JSON.

3.2.7 Completing CDV

After completing the extension of the backend part, it was time to finish our work in CodeDiffVisualizer. Because of our initial research about porting browser extensions from Chrome to Firefox and our initial modifications to CDV, we estimated this to be a more or less straightforward task. Nevertheless, we quickly ran into issues. Due to our new architecture, the opening of the separate graph window was no longer intended to be done by direct user interaction. As mentioned before, CDV should open the graph as soon as it receives the necessary information through native messaging from CDP. What we didn't know was that opening new windows from popups without user interaction is not allowed, at least in Firefox. Since the error messages were hard to interpret, we had multiple lengthy debugging sessions to find out this was the cause. We solved this by opening the graph window directly when the user clicks on the *Start Fetching* button after entering the needed information into the popup. We then update the window's content once CDP has informed CDV that the data to show the graph is ready. To implement this approach, we needed to do a lot of refactoring in CDV. The most changes we had to make were replacing some of the Chrome API functions that use completion blocks with the equivalent promise-based functions of the Firefox API. With these changes, we were able to initially open the window for the graph and update it with possible error messages and other information.

One of the last steps would have been to update the window displaying the graph. Regardless of the many debugging sessions and different approaches where we seemed to come closer to the resolution of the issue, we were still unable to do so in the end. Due to this and the prior mentioned hurdles we had to overcome, time ran short, and we could not resolve all the open issues we wanted to add. The complete list of shortcomings will be discussed in Section 3.4.

3.3 Conclusion of Extension

During our work, we modified both parts of ReviewVis heavily. The backend part was changed by adding a fetcher that retrieves the files from Phabricator. We implemented it in a way that allows adding additional fetchers from other source control management platforms in the future easily. Similarly, we changed the architecture of the parsing, and new parsers for other languages can be integrated straightforwardly during future extensions. Regarding the frontend, the browser extension was modified to enable the user to provide the needed input for fetching the files directly and to allow it to communicate in a bi-directional way with the CDP.

The whole extension process of the tool turned out not as smooth as we hoped it to be. We had to overcome multiple significant obstacles during our work on the modifications. One of those was finding a suitable C++ parser. Finding the proper procedure to determine the needed values to create nodes for our graph was more tedious than expected. Then there were the issues with retrieving the required files from the Phabricator API we mentioned before, which were more complex than anticipated. In addition, the final modifications to CDV, making it display the graph, came with problems we did not anticipate during our planning and could not be completed in time.

Because of those and other minor issues, we could not do the tool's test run with Mozilla during our work on this thesis. We, therefore, decided to miss out on the test run and its evaluation and created a plan to take a step back and do general research on visualization tools. We decided to do so because it would allow us to continue to work on completing a minimal working version of the adapted ReviewVis while carrying out the study next to it. We will present the research we did in Section 5.

3.4 State of ReviewVis

In this subsection, we present a list of the shortcomings our version of the tool currently has compared to the previous versions. Then we will summarize possible future additions to ReviewVis, which originated from our requirements and the previous work of Botschen and Fröhlich.

3.4.1 Shortcomings

During our work on ReviewVis, we initially focused on creating a minimally viable version of it for Mozilla, which we could then build upon during the planned test run. However, as explained before, we could not finish that initial version in a timeframe that would have allowed us to do the test run and evaluate it. We, therefore, decided to continue to work on the tool while also doing research. In the end, we were unable to finish the tool and could not re-add some of the missing features we planned to, to our version. An overview of the features we could not implement can be found below in Table 2.

Title	Component	Description
Graph	CDV	We could not make the graph display using Firefox. The exact origin of the issue is still very much unknown since no obvious error messages are displayed. It is also hard to estimate how long it would take to complete this.
Graph Interaction	CDV	Due to not being able to display the graph, we were also not able to port features like the highlighting of the nodes when hovering over the respective parts of the diff in Phabricator.
Settings	CDV	During refactoring of the CDV, we re-used the settings view to create our input view. We did not find time to add it again in the end. The files containing the code of the settings view are, however, still in the project.
Java parsing	CDP	We did not look at a way to integrate the java parsing into the logic we created, and it is therefore not possible right now. It might be best to call it from our CDP and use the JSON-Combiner by Botschen.

Table 2: Missing features in our version of ReviewVis

3.4.2 Future Work

The goal of this section is to collect possible improvements to ReviewVis. These improvements, listed in Table 3, originate from previous work, ourselves, and Mozilla. The missing support for additional programming languages was not listed intentionally since we see adding them as improvement as self-evident. The improvements we collected from our participants during the interviews will be presented later in Section 6.2.

Title	Source	Description
Renaming	Mozilla & Botschen	Currently, the parser cannot detect renaming changes, leading to such nodes being displayed as deleted and added again.
Tests	Mozilla	Tests are included in the graph as regular nodes. Mozilla was suggested to display them differently and to add a setting to hide changes related to them.
Transitive Dependencies	Fröhlich	Transitive dependencies are currently not visible in the graph if the node in the middle is not contained in the graph.
Comment changes	Fröhlich	Changes that only concern comments are displayed as regular nodes. It was suggested to add a setting to hide such changes or remove them completely.
Show more Information	Botschen	Surveys from previous work showed that developers would appreciate more information directly in the graph, e.g., LOC changed or class size.
Add Color Options	Botschen	Participants in the thesis requested more coloring options, which included a color option for color-blind users.

Table 3: Suggested features for ReviewVis

4 Research questions

The practical part of this thesis aimed to extend and modify ReviewVis to make it possible for Mozilla to use it. To enable this, we first had to adapt the CDV extension to be supported by Firefox. Mozilla also provided us with their three most used languages, which they liked us to add support for. We decided upon implementing a C++ parser into CDP since it is their most used language and, therefore, would benefit them the most. During the extension of the CodeDiffParser, we also modified its internal architecture to facilitate future additions of language parsers.

Next to this adaption of the tool for Mozilla, we wanted to take a step back during the theoretical part. The goal was to confirm that the approach Fröhlich took for the visualization of ReviewVis, is well received among reviewers. To gather this information, we presented ReviewVis together with mockups based on the tools from Section 2.2 to developers during semi-structured interviews. During these interviews, we collected impressions and opinions about the strengths and weaknesses of each tool. This allowed us to compare them and determine what benefits or drawbacks impact the developers the most. In addition, we also wanted to find out if the developers prefer ReviewVis and its graph over the other tools' visualization methods when reviewing merge requests of different sizes. To answer these open points, we created two research questions.

RQ1 Is the graph-based visualization technique ReviewVis uses preferred among developers over different techniques from other tools?

With RQ1, we try to determine if developers would choose ReviewVis over other tools that use different techniques to display additional information during code review. For this, we introduce mockups based on ReviewVis and the three other tools presented in Section 2.2 to our interviewees. To gain insights into how the tools are received and to be able to compare the visualizations of the tools, we ask the participant the same set of questions for each tool. These sets contain questions about possible benefits and drawbacks the participants think the tools could provide to them, but also about possible improvements they could imagine. The complete sets of questions can be found in Appendix A.

RQ2 Do developers still prefer ReviewVis over other options of change visualization tools when it reaches its known limitations regarding the size of the merge request it is used for?

To answer RQ2, we asked our interviewees to select one of the tools to use for hypothetical code reviews they would have to complete. For this, we provided the participants with merge requests having differing characteristics regarding the size of them. We did this because we wanted to focus on the main limitation of ReviewVis, which is that it might not bring as many efficiency gains for larger and smaller merge requests [Fröhlich, 2020]. In this way, we were able to see if they preferred other tools in these cases and possibly find features of them that could be added to ReviewVis to mitigate its limitations.

5 Evaluation

In this section, we present and evaluate the semi-structured interviews we conducted. In Section 5.1, we present the methodology and the interview participants' demographics. Next, we will present the benefits and drawbacks mentioned in the answers of our participants for each tool we showed them. In Section 5.3, we present the results of the comparison of the tools we did during the interviews. Finally, we present the threats to the validity of our work in Section 5.4.

5.1 Methodology

We conducted the interviews in a semi-structured manner. The interview guide was split into four sections. First, in an introductory part, where we asked the participants about their experience in software engineering and code review in general. In the second section, we presented each of the four tools in random order to them. The order of the tools we interviewed each participant in can be found in Table 4. We showed them the printed-out mockups based on the tools from Section 2.2 and gave them a brief introduction to each of these tools. Then we asked the interviewees a set of nine questions for each mockup. We did not tell them we worked on ReviewVis during this thesis to prevent biased answers. In section three of the interview, we wanted to gather information on whether other tools would be preferred in cases where ReviewVis reaches its limitations. The last section of the interview consisted of just a round-up question. The complete questionnaire can be found in Appendix A.

5.1.1 Demographics

We interviewed 6 participants in total. We listed information about them in Table 4, sorted by the date the interview was conducted. We collected all the demographical information in the first section of the interview. All the participants worked at the same company and performed code reviews regularly. The fact that they work in the same company brought the benefit with it that they all use the same SCM tool, which was, in this case, GitLab. Like this, we can use the tool as a common baseline, functionality and feature-wise, and focus only on tools that would be used next to it.

The random tool order, found in Table 4 that each participant was interrogated in, is encoded as follows: 1.) ReviewVis, 2.) ChangeVis, 3.) ViDI, and 4.) Softagram.

ID	Gender	Job Title	Coding Experience in Years	Code Review Experience in Years	Tool Order in Interview
P1	Male	Software Engineer	22	10	2, 1, 3, 4
P2	Male	Software Engineer	14	5	4, 3, 1, 2
P3	Male	Software Engineer	6	3	3, 4, 1, 2
P4	Male	Software Engineer	12	4	3, 2, 4, 1
P5	Male	Software Engineer	15	7	4, 1, 2, 3
P6	Male	Software Engineer	15	10	1, 2, 4, 3

Table 4: Interview participants overview, ordered by date of interview

5.2 Individual Tools

In this subsection, we present the by the participant's mentioned benefits and drawbacks of each tool we showed them individually. The images we used are based on the tools introduced in 2.2 but might have been slightly adapted to be better suited for using them during code review. The mockups we created can be found in Appendix C.

5.2.1 ChangeVis

ChangeVis uses the most textual approach for its visualizations among our selection of tools. The additional information it provides is added by placing the links of references and definitions to both sides of the diff view itself. This makes the dependencies, and through that, also the impact of the changes visible at a glance. All participants [P1, P2, P3, P4, P5, P6] favored this benefit when we showed them the mockup. Another often mentioned benefit was that it allows for navigation directly in the browser, so the developers would not have to open their integrated development environment (IDE) [P1, P2, P3, P4, P6].

On the contrary, the participants also stated some drawbacks of the tool. According to them, the way the links are represented leaves room for improvement. P5 argued that the columns of links would get too crowded if the change block contained many references and that this would increase the possibility of overlooking references of the change that could have a critical impact. P1 mentioned that the additional columns would use even more of the already scarce screen estate available on their machine. To solve these issues, the participants proposed a variety of solutions. The most prominent solution was incorporating the links into the diff view itself. They proposed to directly make the belonging code elements interactive, such that when hovering over them, they would show additional info [P1, P3, P4, P6]. P1 further criticized how the path the links point to is displayed entirely. To solve this and save space, he proposes to combine links that point to the same file. This is in line with the suggestions of P4 and P6, who stated that the text of the links must not be truncated if they are too long. P4 argues that this is too much to process and could also lead to essential info being hidden and thus overlooked.

Similarly to others, P5 proposes to remove the textual links altogether. But instead of moving the info into the diff view, he would replace the links in the columns with items that only show the links when interacting with them. He argues that the items should still somehow display the number of links it contains to provide glanceable information about the number of references and definitions in a block. For this, he proposes to encode this information into the size of the items. Next to these improvements, the participants also stated feature requests. P5 suggested adding the information if tests cover the changed block, and P1 would find it helpful to see if the file the link points to also has some changes in it without actually having to open it. They, however, did not provide suggestions on how to implement these requests.

5.2.2 ViDI

The visualization based on ViDI was the only one that used a city-based visualization technique. The most mentioned benefit by the participants was that the visualization made classes with many changes visible at first sight [P3, P4, P5]. P4 and P5 would find this especially useful to help them decide in which order to review the changes. P5 adds that this might additionally reveal what classes are getting too big and might need refactoring.

The amount of information in the visualization of ViDI was perceived as insufficient by the participants. P1, P2, P4, and P6 think the information provided without interaction is primarily irrelevant for the review itself because it just shows the number of classes and the modified methods contained in them. These participants suggested adding more details, like class or method names, and maybe even LOC changed to the elements textually. P4 and P5 propose using the elements of the visualization themselves to show additional information. They suggest that the height of the cubes could reflect the lines of code changed in the method or even a different measure like complexity. Another thing mentioned by multiple people was the arbitrary positioning of the towers in the graph [P3, P4, P5]. While P5 was uncertain how to solve this, P3 and P4 suggested creating neighborhoods with related classes placed next to each other. This could also help with navigation and orientation when using the graph, which P2 assumed to be difficult without having more reference points. For P5 and P6, the graph also lacks the possibility to show dependencies or references. P6 suggests this could be shown by lines connecting the method-cubes of the tower. He suggests only showing them when hovering over a cube not to create too much visual noise.

P1, P2, and P3 argue that the visualization of ViDI does not display the relevance of the changes in an appropriate manner. P3 even adds that the three-dimensional layout, as it is, is just a gimmick and does not provide any benefits and that the towers could even be displayed in two dimensions. He adds that the ratio of the space the visualization uses versus the information it provides is not excellent and could be improved.

5.2.3 Softagram

Most participants said that the picture based on Softagram could be used to get a quick overview [P2, P3, P4, P5]. In the same take, all participants stated that the visualization looked too dense and crowded. P1 nevertheless liked that it looked structured. The participants gave a lot of suggestions for improvements to the tool. Next to adding more space between the elements, P1 suggested using more prominent colors in the elements and in the graph. P4 suggested adding the ability to track progress on what components have already been reviewed. He also adds it would be beneficial to somehow integrate a change's impact into the graph, for example, by varying the thickness of the arrows between the elements. P5 also mentioned that the impact of changes is not as easy to see. He further adds that for him, the path-relevant boxes around the elements just add noise and no benefit.

5.2.4 ReviewVis

We also asked the participants the same set of questions for the mockup based on ReviewVis. The overall feedback we got was positive. All of the participants liked that it shows the dependencies at first sight in an understandable way. P3 adds that changes that do not have any relations could be easily spotted and investigated. P4 especially liked that it showed connections he could miss when he would just have looked at the code.

Next to the positive feedback, there were also some aspects the participants didn't like or saw room for improvement. P1 and P3 found that the graph would take a lot of space away and that they might need another monitor to work to use it. Interestingly, only P1 and P2 saw issues with big merge requests getting too crowded and confusing, similar to participants from Botschen thesis. Regarding the improvements, the participants mainly saw a need to display more information directly within the graph's elements. P1 and P5, for example, would like to see inheritance between the classes. P3 said that details like lines of code changed could be added so that he doesn't have to look in the diff. Further, for P4, the option to somehow see the imports would be beneficial. But there were also some significant feature requests. P5, for example, would like the possibility of displaying the whole code base using ReviewVis. P2 and P6 suggested improving the distribution of the nodes in the graph such that more related changes are placed closer to each other and complexity would be reduced.

5.3 Tool Comparison

This section will cover the answers we got from the participants in part three of the interview. The goal was to collect information about which tool would be preferred overall, but also for different sizes of merge requests.

5.3.1 Overall Preference

After we introduced each tool separately and interrogated the participants about them, we wanted to find out which of the tools they preferred to use overall and why. P3, P4, P5, and P6 stated that if they had to select one, they would most certainly choose ReviewVis. If it existed, P3 and P6 said they would prefer a combination of ReviewVis and ChangeVis. The reasons why the participants decided to use ReviewVis are very similar to each other. All four participants that favored ReviewVis said they like how it provides a quick and easily understandable overview of the changes contained in the merge request. P4 and P5 specifically chose it because it would provide them with the most information without needing any interaction.

P1 and P2 decided that ChangeVis would suit them the most during their daily work. P2 chose it because it is the least gimmicky, and he likes how its look resembles that of an IDE. P1 likes that it allows him to navigate between files directly on the web and that the links used to do that are presented in the same window as the changes.

Figure 9 below shows the distribution of the participant's choices of which tool they would most likely use.

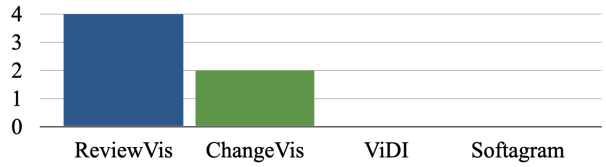


Figure 9: Tool chosen as most likely to be used

On the other hand, we also asked the participants which tool they would least likely see themselves using during their everyday work. Two-thirds of the developers chose the tool based on ViDI [P1, P2, P4, P6]. They all argue that the info of just seeing the size of the merge request depicted by the towers does not help them much during a review.

P3 and P5 chose the tool based on Softagram to be the least likely they would use. P5 argued that it simply looks too complicated compared to the other tools available. P3 chose it because of his previous answers: that it seemed too crowded and complex and that he could use the IDE instead.

The choices the participants made for the tool they would least likely use are shown below in Figure 10.

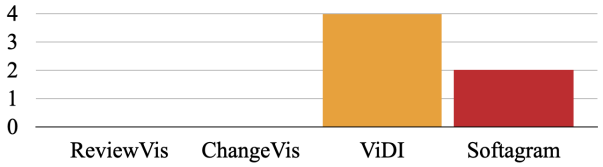


Figure 10: Tool chosen as least likely to be used

5.3.2 Specific Preference

To answer *RQ2*, we asked the participants to choose the tool they would most likely use for reviewing different sizes of merge requests. We knew from previous work by Botschen and Fröhlich that developers stated ReviewVis would lose some of its benefits when using it for smaller and larger merge requests. Because of this, we wanted to determine if they would choose other tools when the visualization used in ReviewVis is used for these sizes of merge requests.

We asked the participants to choose a tool for a small, a medium, and a large merge request. Before the interviews, we, the authors, discussed where to draw the line between small, medium, and large merge requests. We decided that files between four and seven changed files would count as medium-sized, below as small, and above as large. During the first interviews, the participants already disagreed with our classification. So we asked them for their definitions of the sizes. These definitions can be found in Table 5. P6 argues that the size of a merge request depends more on the complexity of the changes than a numerical value like LOC or number files changed. For him, a large merge request can be a single change that has a lot of impact on the logic of the overall codebase, and a small merge request could be a renaming that spans multiple dozens of files.

ID	Small	Medium	Large
Authors	<4 changed Files	4-7 changed Files	> 7 changed Files
P1	< 4 changed Files	4 - 72 changed Files	> 72 changed Files
P2	< 20 LOC	20 - 1000 LOC	> 1000 LOC
P3	< 10 changed Files	10 - 20 changed Files	> 20 changed Files
P4	< 250 LOC	250 - 2000 LOC	> 2000 LOC
P5	< 100 LOC	100 - 500 LOC	> 500 LOC
P6	*Depending on Complexity*	*Depending on Complexity*	*Depending on Complexity*

Table 5: Participants definition on sizes of Merge Requests

For large merge requests, all participants stuck to their initial choice from above; see Figure 11 for the distribution. P5 mentioned ReviewVis would still show him the relations and dependencies, no matter the size. P4 also chose ReviewVis because it provides him with the best overview and helps him to prioritize which changes to look at first. P2 would still use the visualization based on ChangeVis but mentions that ReviewVis could be the better choice for people that prefer visual information.

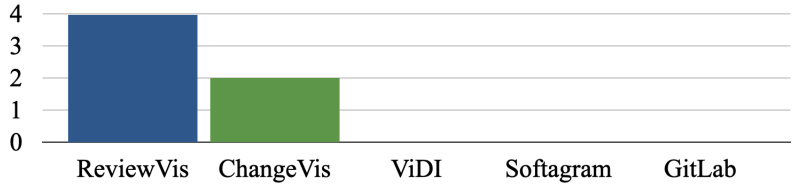


Figure 11: Tool preferred for large merge requests

When asking the participants for their choice, if they had to review a medium-sized merge request, nobody would have changed their tool of choice. See Table 12 for their choices.

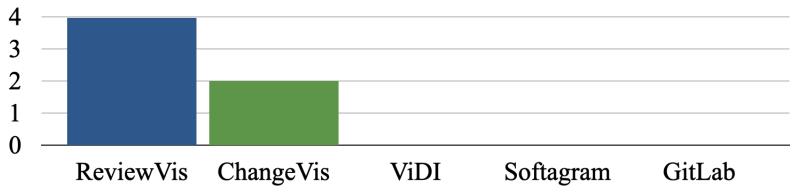


Figure 12: Tool preferred for medium merge requests

At last, we asked them for their choice for reviewing a small request, and some participants changed their tool of choice. P4 and P5 switched from ReviewVis to ChangeVis, and P3 and P6 switched from ReviewVis to just using the GitLab view. P4 and P5 argued that a small number of changes could also be efficiently reviewed with ChangeVis and that the information it provides would be sufficient to see the impact. P3 and P6 said that for them, no tool support would be needed for small merge requests.

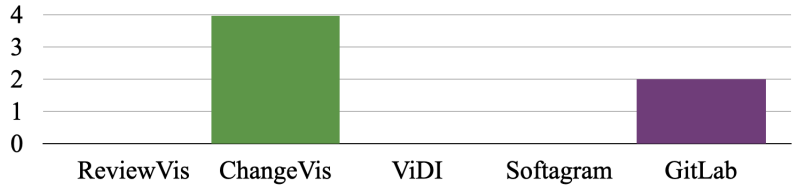


Figure 13: Tool preferred for small merge requests

5.4 Threats to Validity

In this section, we cover both the external and internal threats to the validity of the study we conducted. We present the impact these threats might have had on the study and how we tried to reduce their possible effects.

5.4.1 Threats to Internal Validity

Interviewer effects. The interviewer was also the person that extended the tool ReviewVis. Therefore, the posed questions were specifically designed in an open-ended manner not to influence the participants' opinions. All participants also knew the author personally from work and that he was working on his bachelor's thesis. We did not specifically tell them the topic of the thesis or tell them which of the four tools we extended before the interview to remove a possible bias toward our work.

Participant selection. The author selected the participants by posting a chat message in the company's internal messaging system. To reduce peer pressure, they were asked to reply in a personal chat between them and the author. Additionally, they were told they could drop out of the study anytime they wanted to.

Social Interaction. The participants could voluntarily join the study by replying to the author in a private conversation. This led to them not knowing the other participants without actively trying to find out by asking the other employees directly. The exchange between the participant could, however, not be mitigated with certainty. We still asked them not to talk with their peers about the study after the interview to reduce this chance further.

Tools Prototype. The tools were only presented by showing the participants an image printed on paper. Like this, the participants needed to imagine using the tools instead of actively interacting with them. To help the participants as much as possible, we explained each tool and gave them time to ask questions before posing the interview questions.

5.4.2 Threats to External Validity

Sampling Bias. All six participants were recruited from the same company. All the reviews performed in all teams use Gitlab as the main tool when doing reviews. Furthermore, all participants have been conducting code reviews regularly for multiple years. Therefore, we argue that the selected participants were suited to give insights relevant and applicable to developers in general.

Experimenter effect. The interviews were conducted by the author of this thesis. The conductor was also an employee of the company the participant came from. To mitigate the possible influence this fact could have on the interviews, the guide was thoroughly reviewed by a person familiar with conducting interviews, and the interviewer was introduced to the matter of doing them by this person.

6 Discussion

In this chapter, we will discuss the insights we gathered from the replies we got during the semi-structured interview and answer the research questions we posed. We will further present a list of possible improvements to ReviewVis brought up by the participants of the interviews.

6.1 Research Questions

RQ1 *Is the graph-based visualization technique ReviewVis uses preferred among developers over different techniques from other tools?* As presented in Section 5, four out of the six participants would choose ReviewVis over the other tools. These participants primarily chose it because of the initial overview it gives for reviewing. They like that relations are visible at first sight [P3, P4] and that the interactive elements could help them prioritize the files to create an order to review them in [P3, P4]. These are points the participants often missed in the other tools. Especially the tool based on Softagram was criticized because the participants perceived its way of visualization as too dense [P1, P2, P3, P4, P5, P6] and complex at first sight [P2, P4]. On the other side, they found the visualization based on ViDI to give too little [P1, P2, P4, P6] and irrelevant [P1, P2, P3] additional information to be as useful as ReviewVis during reviews. Some further benefits mentioned of ReviewVis are the visibility of internal structures that is harder to find out when just looking at the diff in GitLab [P6]. None of the participants mentioned this benefit existed in other tools.

For two of the six participants, ChangeVis was preferred over ReviewVis [P1, P2]. They argue that ReviewVis, and also some of the other visualizations, are just too playful and gimmicky for them. Nevertheless, they both liked the way they could see how the classes are related in ReviewVis. P2 specifically mentioned that he is not a person who prefers the information to be visualized. Interestingly these participants also were the only ones that saw issues using it when the merge requests would get too big.

Overall, the participants reacted positively to ReviewVis, which is shown by four out of six participants choosing it as their most likely tool to use. The other two participants liked the textual visualization of the mockup based on ChangeVis the most. From their answers to the questions about ReviewVis, we conclude that this is not because they perceive it as a bad visualization but rather because they prefer a visualization technique more similar to the one they are used to from an IDE.

RQ2 *Do developers still prefer ReviewVis over other options of change visualization tools when it reaches its known limitations regarding the size of the merge request it is used for* We knew from the work done by Fröhlich and Botschen that for some users, ReviewVis would become too complex if it is used with large merge requests. Others argued that the usefulness increases with the size of the merge request. We can see a similar distribution from the answers we received during our work. Of the four participants choosing ReviewVis for large merge requests, two explicitly would do so because it gives them an overview of the changes and their relations. Interestingly, only the participants choosing the ChangeVis-based tool had apprehensions that large merge requests might lead to the graph getting too complex.

When asking the participants about smaller merge requests, two of the participants who preferred ReviewVis would switch to using no support tool at all. This is also in line with findings from before. However, it is essential to consider their definition of small merge requests. P3 defines small merge requests as ten files with changes or lower, while P6 relies on complexity to

determine the size of a merge request. As displayed in 5, we see that the definitions of what a small merge request is, are very dependent on the person you ask. Therefore, we argue that it is similar to RQ1 and that it's the personal preference and not the traits of the graph that make the difference between using ReviewVis or not for smaller merge requests.

6.2 Future Work

In this subsection, we summarize the findings we received during phase two of the interview, where we asked the participants about the tools individually. These suggestions mainly stem from the question directly asked about ReviewVis. Still, we also thought about how the most liked aspects of the other tools could be integrated into ReviewVis to increase its usefulness and usability. Together with Table 3 from Section 3.4.2, they should serve as a guide for features to be added in future expansions on ReviewVis.

Title	Origin	Description
Inheritance	P1, P5	ReviewVis has the ability to show inner classes. It would be great also to have a way to represent class inheritance.
Node Distribution	P2, P6	Improve the distribution of the nodes in the graph to reduce complexity.
Clustering	P2	It was proposed to cluster the nodes with related changes in proximity to each other. This would facilitate creating a reviewing order.
Make Nodes Transparent	P3	Instead of deleting the node, simply make it transparent to keep the context.
Additional Information	P3	Add additional information to the graph, like LOC changes, references, and definitions. This could be achieved by making the labels of the nodes show info when hovering over them.
References	P4	Ability to display all references of nodes to see the impact of a change.
Raw Look	P6	P6 mentioned the visualization still looks like a proof of concept and would need an overhaul.
Open Links in Browser	ChangeVis	In ChangeVis, a click on the link opens the file in a separate window. This could be incorporated into ReviewVis by making the class names in the bubbles also clickable.
Hide Reference Connections	ViDI	To reduce the complexity of the graph, a toggle that hides the reference lines could be added to the settings.

Table 6: Features and improvements based on inputs from interview Participants

7 Conclusion

The goals of the practical part of this thesis were to extend ReviewVis to make it usable for Mozilla, make future extensions of it easier in general, and reduce the manual steps users need to do when using it. For this, we first adapted the existing CodeDiffVisualizer Chrome extension to make it run in Firefox. We then added Native Messaging to the tool, which allowed us to communicate between the front and backend parts. We then added a fetcher that retrieves the files belonging to a differential from Phabricator, the source code management platform used by Mozilla. We added a basic C++ parser to the backend to further increase its usefulness to them since it's their most used language. During the work on the parser, we refactored multiple parts of it to make it easier to add more programming language parsers in the future. We had to overcome numerous hurdles during the modification process, and in the end, time ran short. Instead of conducting the trial of the adapted tool with Mozilla, we moved to plan B, described in the next paragraph. This allowed us to, at the same time, continue working on finishing the tool. In the end, we were not able to complete the frontend part of the adapted tool due to issues that arose when adapting the Firefox extension to our new architecture and functionalities.

Instead of the anticipated test run with Mozilla, we took a step back. We tried to find out more about how developers perceive ReviewVis compared to other tools that also create visualizations of merge requests. We presented a mockup based on ReviewVis and three other existing tools to six participants in semi-structured interviews. The first goal was to find out if they would prefer ReviewVis overall and if the other tools have features that ReviewVis could benefit from. We observed that for some developers, a graphical visualization might not be favored and that they prefer a more textual representation of the changes. Nevertheless, four of the six participants chose ReviewVis as the tool they would most likely use. They chose it mainly because of the way it presents them with a graphical overview that helps them understand the changes. We also wanted to determine if the participants would fall back on a different tool in cases where we knew ReviewVis would reach its limitations. For large and medium merge requests, all of the four participants would stick to ReviewVis. Only for small requests, two participants said they would change to not using a tool at all, which is in line with the findings of previous work by Fröhlich and Botschen.

To conclude, we can say that the visualization ReviewVis uses might not be optimal for developers that do not favor visual support tools. But for those that do, it provides a good level of support, which might be built upon by adding the missing features and improvements they

References

Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712–721, 2013. doi: 10.1109/ICSE.2013.6606617.

M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976. doi: 10.1147/sj.153.0182.

Peter C. Rigby, Daniel M. German, Laura Cowen, and Margaret-Anne Storey. Peer review on open-source software projects: Parameters, statistical models, and theory. *ACM Trans. Softw. Eng. Methodol.*, 23(4), sep 2014. ISSN 1049-331X. doi: 10.1145/2594458. URL `https://doi.org/10.1145/2594458`.

Eduardo Witter dos Santos and Ingrid Nunes. Investigating the effectiveness of peer code review in distributed software development. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES'17, page 84–93, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450353267. doi: 10.1145/3131151.3131161. URL `https://doi.org/10.1145/3131151.3131161`.

Vipin Balachandran. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 931–940, 2013. doi: 10.1109/ICSE.2013.6606642.

Emma Söderberg, Luke Church, Jürgen Börstler, Diederick Niehorster, and Christofer Rydenfält. Understanding the experience of code review: Misalignments, attention, and units of analysis. In *The International Conference on Evaluation and Assessment in Software Engineering 2022*, EASE 2022, page 170–179, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450396134. doi: 10.1145/3530019.3530037. URL `https://doi.org/10.1145/3530019.3530037`.

Devarshi Singh, Varun Ramachandra Sekar, Kathryn T. Stolee, and Brittany Johnson. Evaluating how static analysis tools can reduce code review effort. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 101–105, 2017. doi: 10.1109/VLHCC.2017.8103456.

Tobias Baum, Kurt Schneider, and Alberto Bacchelli. On the optimal order of reading source code changes for review. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 329–340, 2017. doi: 10.1109/ICSME.2017.28.

Martin Dias, Alberto Bacchelli, Georgios Gousios, Damien Cassou, and Stéphane Ducasse. Untangling fine-grained code changes. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 341–350, 2015.

Jason Cohen, Steven Teleki, and Eric Brown. *Best kept secrets of peer code review*. Smart Bear Incorporated, 2006.

Lorenzo Gasparini, Enrico Fregnan, Larissa Braz, Tobias Baum, and Alberto Bacchelli. Changeviz: Enhancing the github pull request interface with method call information. In *2021 Working Conference on Software Visualization (VISSOFT)*, pages 115–119, 2021. doi: 10.1109/VISSOFT52517.2021.00022.

Yuriy Tymchuk, Andrea Mocci, and Michele Lanza. Code review: Veni, vidi, vici. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 151–160, 2015. doi: 10.1109/SANER.2015.7081825.

Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: A controlled experiment. pages 551–560, 05 2011. doi: 10.1145/1985793.1985868.

Josua Fröhlich. Code review visualizations with codediffvis for java. Master's thesis, 2020.

Raffael Botschen. Improving codediffvis for code review visualizations. bachelor's thesis, 2022.

Eli Bendersky. Parsing c++ in python with clang, Jul 2011. URL `https://eli.thegreenplace.net/2011/07/03/parsing-c-in-python-with-clang`.

A Interview Guide

Interview X: Code-Review Support-Tools

Section 1:

General:
0.1: How long have you been a developer?

0.2: How long have you been doing code reviews?

0.3: How many times are you doing code reviews a week?

How much time does this take?

Experience with tools:
1.0: Which tools do you use for code review?

1.1: Do you use any supporting tools next to standard diff viewers in your reviews?

1.1.1: No: Did you use any?

Why did you stop?

1.1.2: Yes: What are they called?

How do they support you?

What benefits do you like most about them?

1.2: Have you heard of (other) tools that support engineers during review?

-> Tell them what visualization means: Using an image or graph to convey an object or information in a graphical way.

1.3: What do you think about the idea of using some tool that uses visualization to support code review?

What immediate benefits come to your mind?

What drawbacks?

Section 2:
We first present the tool, by showing the developer images.

-> Shuffle between interviews

Tool 1- ChangeVis:
2.1: What do you think of the tool?

2.2: Do you think the additional info the tool shows is relevant?

2.3: How would the information benefit you during a review?

2.4: How could the displaying of information be improved?

2.5: What possible benefits can you imagine when using the tool?

2.6: Why and how would they benefit you?

2.7: Do you see any drawbacks using the tool?

2.8: Why are they negative, how can this be removed?

2.9: Would you use this tool in combination with the tool you normally use for reviews or just the original one? Why? Why not?

Tool 2 - ViDI:
2.1: What do you think of the tool?

2.2: Do you think the additional info the tool shows is relevant?

2.3: How would the information benefit you during a review?

2.4: How could the displaying of information be improved?

2.5: What possible benefits can you imagine when using the tool?

2.6: Why and how would they benefit you?

2.7: Do you see any drawbacks us ing the tool?

2.8: Why are they negative, how can this be removed?

2.9: Would you use this tool in combination with the tool you normally use for reviews or just the original one? Why? Why not?

Tool 3 - CDV:
2.1: What do you think of the tool?

2.2: Do you think the additional info the tool shows is relevant?

2.3: How would the information benefit you during a review?

2.4: How could the displaying of information be improved?

2.5: What possible benefits can you imagine when using the tool?

2.6: Why and how would they benefit you?

2.7: Do you see any drawbacks using the tool?

2.8: Why are they negative, how can this be removed?

2.9: Would you use this tool in combination with the tool you normally use for reviews or just the original one? Why? Why not?

Tool 4 - Code Bubble/Softagram:
2.1: What do you think of the tool?

2.2: Do you think the additional info the tool shows is relevant?

2.3: How would the information benefit you during a review?

2.4: How could the displaying of information be improved?

2.5: What possible benefits can you imagine when using the tool?

2.6: Why and how would they benefit you?

2.7: Do you see any drawbacks using the tool?

2.8: Why are they negative, how can this be removed?

2.9: Would you use this tool in combination with the tool you normally use for reviews or just the original one? Why? Why not?

Section 3:

Comparison:
3.1: Which of the presented tools would you most likely use?

3.2: Why?

3.3: Which tool the least likely?

3.4: Why?

3.5: Imagine a large MR 8+ files, which tool would you prefer and why, which feature/aspect does make the difference that led to its selection?

3.6: What about medium-sized, 4-7 files, which tool would you prefer?

3.7: What about smaller, <4 files, which tool would you prefer?

3.8: Why did you choose the same? Why did you change your selection, which aspects made you change?

Section 4:

End:
4.0: Would you like to add any final remarks?

B Interview Consent Form

Zurich Empirical Software Engineering Team (ZEST) University of Zurich
Department of Informatics
Binzmühlestrasse 14
CH-8050 Zurich
Switzerland

Contact Person
Marc Kramer
marc.kramer@uzh.ch

Interview - Participant Consent Form

You are invited to participate in a research interview that is being conducted by Marc Kramer.

Marc Kramer is writing his bachelors thesis in the ZEST at the Department of Informatics, University of Zurich. This work is supervised by Prof. Dr. Alberto Bacchelli, who can be contacted at BACCHELLI@IFI.UZH.CH.

Purpose

The purpose of this research interview is to collect feedback based on the participants' experience with code review support tools.

Study Procedure and Collected Data

If you consent to voluntarily participate in this research interview, your participation will include recording the interview.

Potential Risks

In case you perceive your participation in this study as stressful, you have the right to terminate your participation or to not answer a question. In case of further questions or doubts, you may contact any of the researchers (contacts are provided above) or the OEC Human Subjects Committee of the University of Zurich at HUMAN.SUBJECTS@OEC.UZH.CH.

Conditions of the Study

- **Voluntary Participation.** Your participation in this research must be completely voluntary. Even if you do decide to participate, you can withdraw at any time without any explanation. If you do withdraw from the study, your data will be erased and will not be included in any analysis.
- **Possibility not to answer or withdraw details from your interview.** You are free to not answer all of the questions we ask you. Also, at any moment, you can ask us to remove a specific information from your answers.
- **Confidentiality.** The research staff will protect your personal information closely so no one will be able to connect your responses and any other information that identifies you. Official investigations may require us to show information to university or government officials (or sponsors), who are responsible for monitoring the safety and procedure of our research. Directly identifying information (e.g., names, addresses) will be kept strictly separate at all times from any other collected data and will be stored in a different location. Furthermore, it will not be associated with the data after it has been analyzed. In particular, your contacts, this approved consent, and the recordings of your interview will be securely stored at the University of Zurich in an encrypted archive that only the research staff of ZEST can open. You will not be identified in any publication that derives information from your research interview.

- **Dissemination of Data and Results.** It is anticipated that the analysis of this research interview will lead to results that will be shared publicly (e.g., in form of an academic publication in research conferences or journals). To allow for the reproducibility and reuse of our research, we ask your permission to also share an electronic version of (parts of) your transcribed interview data with the broad scientific community. Any personal information that could identify you will be removed or changed before files are shared with other researchers or results are made public. Data presented in presentations or publications will never allow identifying individual persons.
- **Your personal information.** In case you would be interested and willing to participate in future studies, we will put record of your contacts in a database (different than the one used for the interview data) securely stored at the University of Zurich in an encrypted archive that only the research staff of ZEST can open.
- **Possibility to withdraw data.** If, at any point, you decide that you would like your data to be completely withdrawn from the study, you are free to do so.

Contact for Information about the Study

If you have any questions or desire further information with respect to the study, you may contact Marc Kramer (MARC.KRAMER@UZH.CH) or Prof. Dr. Alberto Bacchelli (BACCHELLI@IFI.UZH.CH).

Consent for Study Participation

Your participation in this study is entirely voluntary. You are free to withdraw your participation at any point during the study, without giving any reason and without any negative consequence. Any information you contribute up to your withdrawal will be retained and used in this study, unless you request otherwise.

With your signature on this form you confirm the following statements:

- I understand the goal and procedures of the study and the applicable conditions.
- I had the opportunity to ask questions. I understood the answers and accept them.
- I am at least 18 years old.
- I had enough time to make the decision to participate and I agree to the participation.

In no way does this waive your legal rights or release the investigators or involved institutions from their legal or professional responsibilities.

Additional consent

[ ] I consent to be recorded during the interview
[ ] I consent to publicly share my anonymized data
[ ] I would like to be contacted for future studies
[ ] I would like to receive the final publications based on my interview data

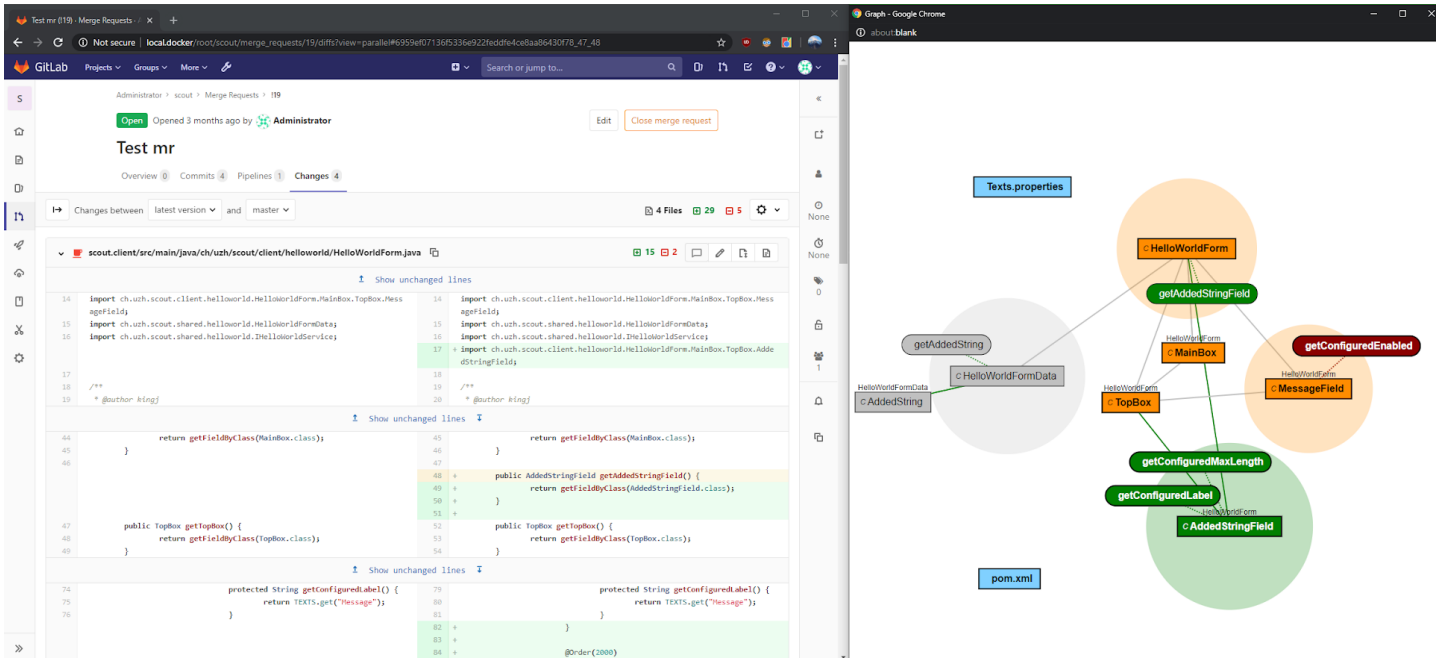
A copy of this consent will be left with you, and a copy will be taken by the researcher.

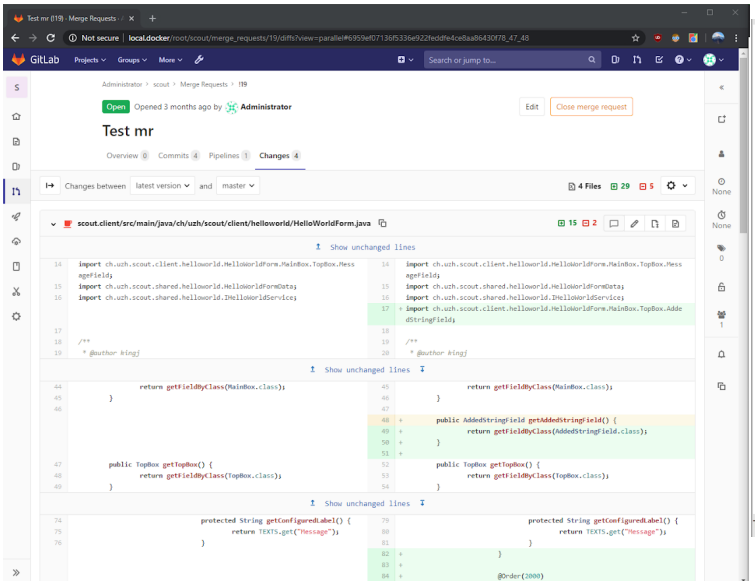
Name of Participant:

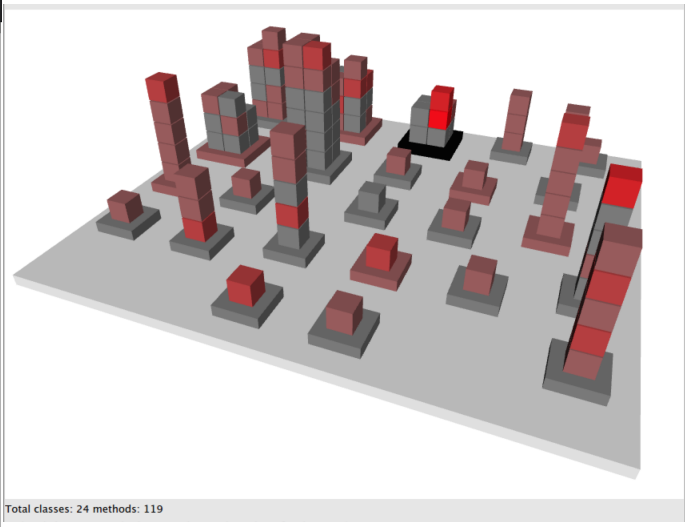
Location, Date:

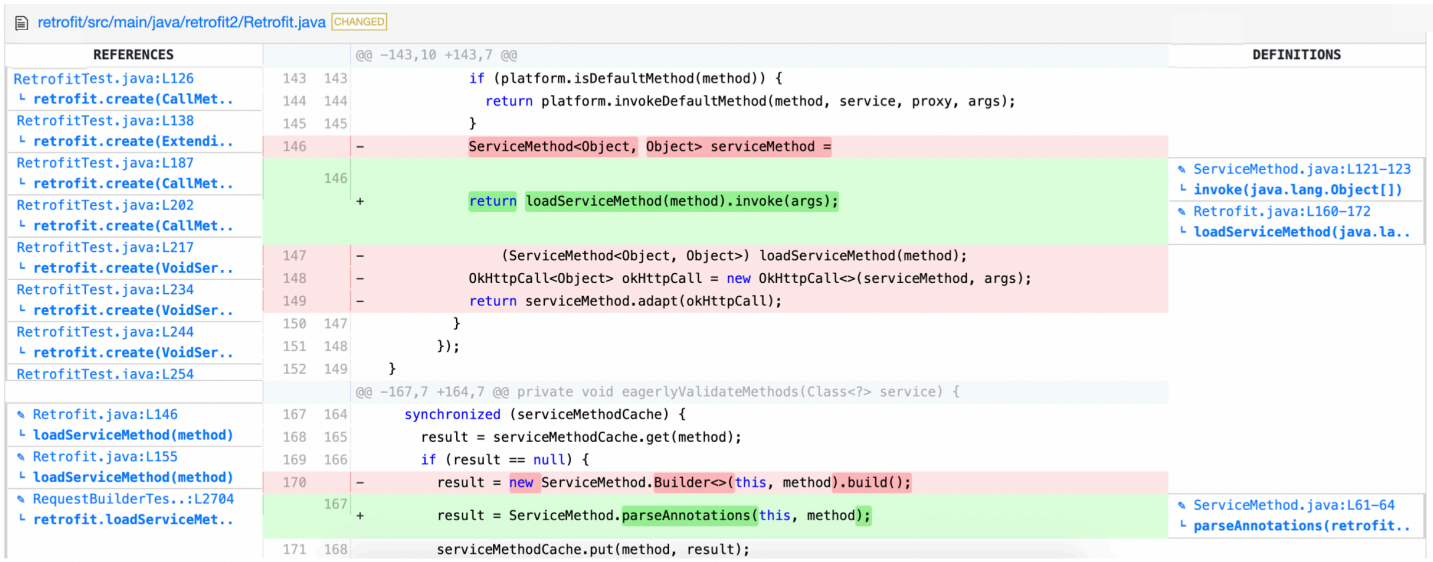
Signature of Participant:

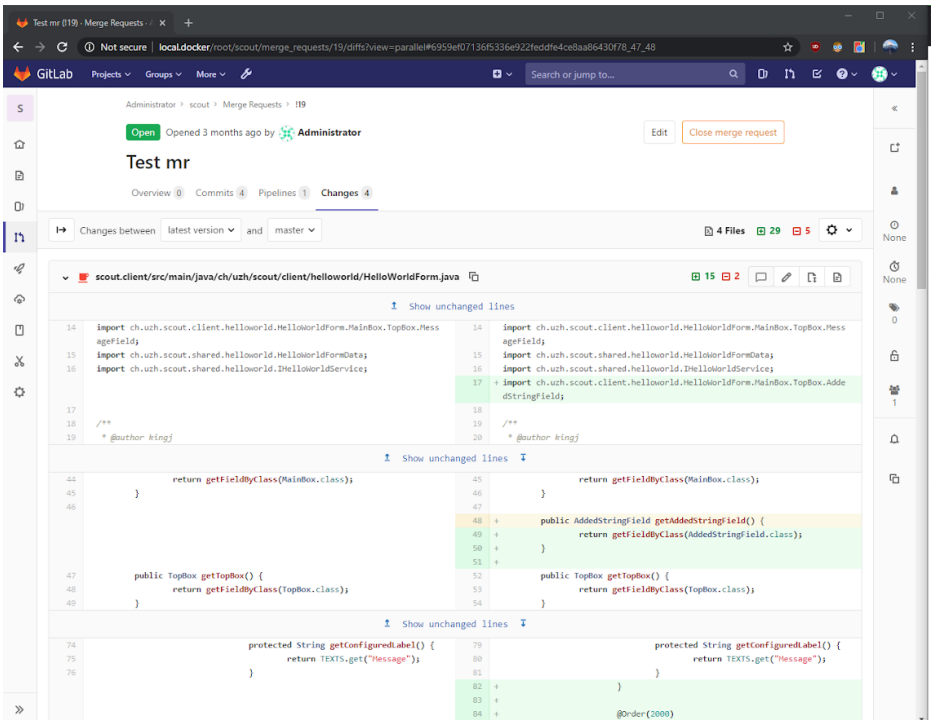
C Interview Tool Images

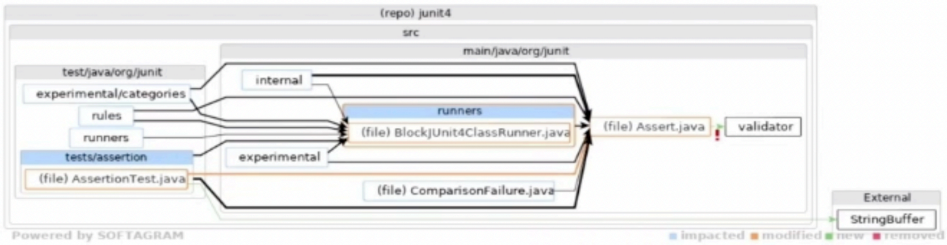

Test mr (119) · Merge Requests
Not secure
local.docker/root/scout/merge_requests/19/diffs?view=parallel#6959ef07136f5336e922feddfe4ce8aa86430f78_47_48
GitLab
Projects
Groups
More
Search or jump to...
Administrator > scout > Merge Requests > 119
Open
Opened 3 months ago by Administrator
Edit
Close merge request
Test mr
Overview 0
Commits 4
Pipelines 1
Changes 4
Changes between
latest version
and
master
4 Files
29
5
scout.client/src/main/java/ch/uzh/scout/client/helloworld/HelloWorldForm.java
15
2
Show unchanged lines
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.MessageField;
import ch.uzh.scout.shared.helloworld.HelloWorldFormData;
import ch.uzh.scout.shared.helloworld.IHelloWorldService;
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.AddedStringField;
/**
* @author kingj
return getFieldByClass(MainBox.class);
}
public AddedStringField getAddedStringField() {
return getFieldByClass(AddedStringField.class);
}
public TopBox getTopBox() {
return getFieldByClass(TopBox.class);
}
protected String getConfiguredLabel() {
return TEXTS.get("Message");
}
@Order(2000)
Graph - Google Chrome
about:blank
Texts.properties
HelloWorldForm
getAddedStringField
getAddedString
HelloWorldFormData
MainBox
HelloWorldForm
getConfiguredEnabled
HelloWorldFormData
AddedString
HelloWorldForm
TopBox
HelloWorldForm
MessageField
getConfiguredMaxLength
getConfiguredLabel
HelloWorldForm
AddedStringField
pom.xml


Test mr (!19) · Merge Requests
Not secure
local.docker/root/scout/merge_requests/19/diffs?view=parallel#6959ef071361f5336e922fedd fe4ce8aa8b430f78_47_48
GitLab
Projects
Groups
More
Search or jump to...
Administrator > scout > Merge Requests > !19
Open
Opened 3 months ago by Administrator
Edit
Close merge request
Test mr
Overview 0
Commits 4
Pipelines 1
Changes 4
Changes between
latest version
and
master
4 Files
29
5
scout.client/src/main/java/ch/uzh/scout/client/helloworld/HelloWorldForm.java
15
2
Show unchanged lines
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.MessageField;
import ch.uzh.scout.shared.helloworld.HelloWorldFormData;
import ch.uzh.scout.shared.helloworld.IHelloWorldService;
/**
* @author hingj
return getFieldByClass(MainBox.class);
}
public TopBox getTopBox() {
return getFieldByClass(TopBox.class);
}
protected String getConfiguredLabel() {
return TEXTS.get("Message");
}
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.MessageField;
import ch.uzh.scout.shared.helloworld.HelloWorldFormData;
import ch.uzh.scout.shared.helloworld.IHelloWorldService;
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.AddedStringField;
/**
* @author hingj
return getFieldByClass(MainBox.class);
}
public AddedStringField getAddedStringField() {
return getFieldByClass(AddedStringField.class);
}
public TopBox getTopBox() {
return getFieldByClass(TopBox.class);
}
protected String getConfiguredLabel() {
return TEXTS.get("Message");
}
@Order(2000)


Total classes: 24 methods: 119


retrofit/src/main/java/retrofit2/Retrofit.java CHANGED
REFERENCES
DEFINITIONS
@@ -143,10 +143,7 @@
if (platform.isDefaultMethod(method)) {
return platform.invokeDefaultMethod(method, service, proxy, args);
}
- ServiceMethod<Object, Object> serviceMethod =
+ return loadServiceMethod(method).invoke(args);
- (ServiceMethod<Object, Object>) loadServiceMethod(method);
- OkHttpCall<Object> okHttpCall = new OkHttpCall<>(serviceMethod, args);
- return serviceMethod.adapt(okHttpCall);
}
});
}
@@ -167,7 +164,7 @@ private void eagerlyValidateMethods(Class<?> service) {
result = serviceMethodCache.get(method);
if (result == null) {
- result = new ServiceMethod.Builder<>(this, method).build();
+ result = ServiceMethod.parseAnnotations(this, method);
serviceMethodCache.put(method, result);
RetrofitTest.java:L126
retrofit.create(CallMet..
RetrofitTest.java:L138
retrofit.create(Extendi..
RetrofitTest.java:L187
retrofit.create(CallMet..
RetrofitTest.java:L202
retrofit.create(CallMet..
RetrofitTest.java:L217
retrofit.create(VoidSer..
RetrofitTest.java:L234
retrofit.create(VoidSer..
RetrofitTest.java:L244
retrofit.create(VoidSer..
RetrofitTest.java:L254
Retrofit.java:L146
loadServiceMethod(method)
Retrofit.java:L155
loadServiceMethod(method)
RequestBuilderTes..:L2704
retrofit.loadServiceMet..
ServiceMethod.java:L121-123
invoke(java.lang.Object[])
Retrofit.java:L160-172
loadServiceMethod(java.la..
ServiceMethod.java:L61-64
parseAnnotations(retrofit..


Test mr (!19) · Merge Requests
Not secure | local.docker/root/scout/merge_requests/19/diffs?view=parallel#6959ef07136f5336e922feddfe4ce8aa86430f78_47_48
GitLab
Projects
Groups
More
Search or jump to...
Administrator › scout › Merge Requests › !19
Open
Opened 3 months ago by Administrator
Edit
Close merge request
Test mr
Overview 0
Commits 4
Pipelines 1
Changes 4
Changes between
latest version
and
master
4 Files
29
5
scout.client/src/main/java/ch/uzh/scout/client/helloworld/HelloWorldForm.java
15
2
Show unchanged lines
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.MessageField;
import ch.uzh.scout.shared.helloworld.HelloWorldFormData;
import ch.uzh.scout.shared.helloworld.IHelloWorldService;
import ch.uzh.scout.client.helloworld.HelloWorldForm.MainBox.TopBox.AddedStringField;
/**
* @author kingj
return getFieldByClass(MainBox.class);
public AddedStringField getAddedStringField() {
return getFieldByClass(AddedStringField.class);
public TopBox getTopBox() {
return getFieldByClass(TopBox.class);
protected String getConfiguredLabel() {
return TEXTS.get("Message");
@Order(2000)


(repo) junit4
src
main/java/org/junit
test/java/org/junit
experimental/categories
rules
runners
tests/assertion
(file) AssertionTest.java
internal
runners
(file) BlockJUnit4ClassRunner.java
experimental
(file) ComparisonFailure.java
(file) Assert.java
validator
External
StringBuffer
Powered by SOFTAGRAM
impacted
modified
new
removed