The Role of Commit Messages in Hierarchy-Oriented Visualization

Dawid Wozniak
IT University Copenhagen
2023
Copenhagen, Denmark
dawo@itu.dk

***Abstract*—The extracting and composing insights from commit messages are particularly complex when one needs to do it in a framework and commit convention agnostic way. Therefore, the commit messages were not often used to enrich the project visualisation in hierarchy-oriented visualization tools. Its role was undefined. To address this problem, we chose one of those tools called Git Truck and developed its new version which includes the commit history visual representation and interactions with it. They are used to enhance the overall project visualization. The new functionalities were derived through a few iterations with stakeholders. Finally, we conducted a preliminary user evaluation with 22 participants who confirmed the usefulness of the new version. Based on the feedback, we defined the role of commit messages in hierarchy-oriented visualization.**

I. Introduction

GIT version control system is the most popular tool among developers [1]. They are many tools to visualize projects using information based on the GIT version control system [2] [11]. One of them is GitTruck [3] [4]. It provides information about the projects based on commits, however, the current version does not show insights from commit messages. The main challenge preventing authors from doing it is exploring the role, desired representations, interactions with them, and assessing the usefulness of the commit messages view in GitTruck.

There are already tools to provide stakeholders with insights using commit messages. There are tools to identify refactoring opportunities by analyzing developer commits and focusing on their messages. [5]. Another approach to commit messages can be seen as a text classification problem [6] [7] [8]. However, those tools provide no or only very technical way to access presented mechanisms that are not friendly for non-technical stakeholders. It might be considered as an extra barrier and accessibility issue [10]. There are also tools to visualize and analyze the repository. Besides mentioned GitTruck [4], SolidTA, Gource, GitStats and CodeFlower do not use commit messages to enrich the project visualisation [2], and some are presented without assessment with the users [11].

In this situation, we decided it is beneficial to investigate which insights, representations and interactions with commit messages are useful for stakeholders and can be supported in the hierarchy-oriented visualization tools. Therefore, we formulate the research question in the following manner: *Which insights about commit messages and what kind of representations and interactions with them are useful for stakeholders when exploring a project using a hierarchy-oriented visualization tool?*

To address this question, we develop a new commit detail tab that enables us to represent and dive into commit messages. It is also a place of new interactions between them and other parts of the visualisation.

II. Methodology

Our methodology to define which role, interactions and insights of commit messages are desired by stakeholders was a two-levels evaluation with its potential users.

Feedback was collected during meetings in the one-to-one formula. They were documented by transcriptions. All transcriptions are attached to this paper as appendixes. The complete list of them can be found in Section VIII.

All our testers can be split into one of two groups:

- Multi-Stage evaluation testers
- Final evaluation testers

The tool has been presented to a tester during a meeting but they could also try it by themselves by running a command *npx -y git-truck-beta@latest* on their machines. If they would like to see the project code, it was also available to them [12].

Multi-stage evolution tester is a person who was interviewed by us four times. Firstly, we showed them the product as it was. We asked what they would like to have in a hierarchy-oriented visualization tool like Git Truck in the context of commit messages. We asked especially about commit messages' role, importance and potential applicable interactions. We repeated this procedure four times, each time adding questions regarding implemented improvements based on the previous feedback iteration.

Final evaluation tester is a person who was interviewed by us once. We showed them the final product including our changes. We asked them about their opinion about our changes, the usefulness of the tool and potential missing features in the context of commit messages.

Regardless of the group, a tester belongs to, they were asked those questions:

(i) For how many years have you worked professionally with IT projects (including part-time jobs)?
(ii) What is your primary role?

(iii) How big is on average your team including all different roles (developers, QAs, PMs, POs)?
(iv) How many commits are circa in your average project?
(v) How many files are there in your average project?
(vi) Would you like to be informed about the project's results?

A. Multi-stage evaluation testers

During our four meetings, we have asked various questions regarding the product overview, their opinions on the commit messages role, and our implementations at the given stage. Each interaction was aimed at attending to the individual feedback provided by our testers; nonetheless, we took the initiative to compile a comprehensive set of questions that were posed uniformly to all participants. The total number of stage testers is 9. They are marked as [P1], [P2], [P3], [P4], [P5], [P6], [P7], [P8] and [P9]. All stage testers participated in all four iterations.

Our questions to testers are listed below.

1) First iteration:

(i) To what extent on a scale of zero to ten, do you agree with the statement "Commit messages are important to me"?
(ii) To what extent on a scale of zero to ten, do you agree with the statement "Information about the commit messages is presented in a clear and easy-to-navigate UI in GitTruck"?
(iii) To what extent on a scale of zero to ten, do you agree with the statement "I believe it would be beneficial to use GitTruck in my work (any purpose)"?
(iv) What would be your main goal to use GitTruck?
(v) Is GitTruck missing anything related to the commit history?
(vi) If you had unlimited resources, time and money, what would be the most useful insights about commits, especially commit messages?
(vii) Is there anything else you would like to say about GitTruck?

2) Second iteration:

(i) What do you think about the new tab view for commits? Is it better than having this view in the *General* tab?
(ii) Would the tag cloud based on commit messages be valuable for you?
(iii) Is the current filtering useful for you? Why?
(iv) What can still be improved to get more insights from commit history, especially commit messages?
(v) Is there anything else you would like to say about the product?

3) Third iteration:

(i) How do you like the UI of *commit details* tab?
(ii) If you had unlimited resources, time and money, what would be the most useful insights about commits, especially commit messages?
(iii) Would you like to have any other filters or sorting options regarding commit messages?
(iv) What do you think about the commit changes visualisation?
(v) What do you think about the idea of listing files frequently changing with a selected object?
(vi) Is the commit size important to you to know?
(vii) What would be your primary usage of this commit view?
(viii) Is there anything else you'd like to say about the product?

4) Fourth iteration:

(i) Has the product improved? How much on the scale from zero (not changed) to ten (fundamental change)?
(ii) To what extent on a scale of zero to ten, do you agree with the statement "Information about the commit messages is presented in a clear and easy-to-navigate UI in GitTruck"?
(iii) How much would it be beneficial to use GitTruck in your work (any purpose) on a scale of zero to ten? Why?
(iv) What is still missing regarding commit message history?
(v) Is there anything else you'd like to say about the product?

B. Final evaluation with new testers

During our one meeting, we presented the final product to our testers. The total number of evaluation testers is 13. They are marked as [P10], [P11], [P12], [P13], [P14], [P15], [P16], [P17], [P18], [P19], [P20], [P21] and [P22].

We asked them the following questions:

(i) To what extent on a scale of 0 to 10, do you agree with the statement "Information about the commit messages is presented in a clear and easy-to-navigate UI in GitTruck"?
(ii) How much beneficial would it be to use GitTruck in your work (any purpose) on a scale of 0 to 10? Why?
(iii) What would be your primary usage of this commit view?
(iv) What is still missing regarding commit message history?
(v) Is there anything else you'd like to say about the product?

III. Results

A. Final product

We present a new representation of commit history included in the detail section for files and folders. Figure 1 shows the user experience after clicking an object in the visualisation.

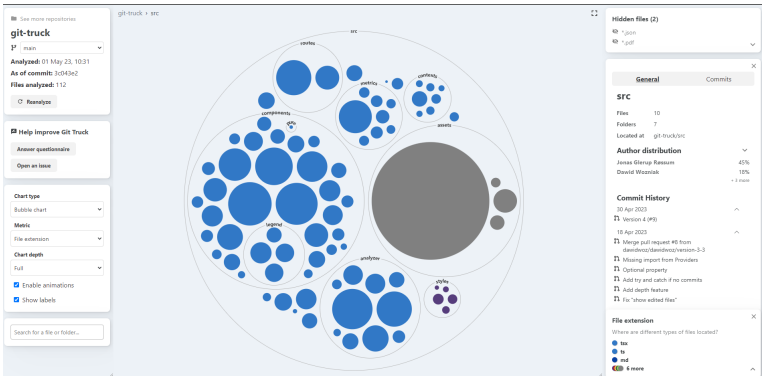

Fig. 1. Overview after clicking an object

The commit history section can be accessed in two ways: limited view in *General* tab and detailed view in *Commits*

tab. Commits within the repository are organized and grouped based on the day they were pushed, providing a chronological arrangement that enables users to see the progress of changes and track the development timeline more effectively. Figure 2 shows an example group of commits. On the top of the group, we placed the date in the format *dd mmm yyyy*. Every single commit is represented by an icon followed by the commit message. The list of commits is ordered from the latest to the oldest on the given day. By default, the group is displayed in an expanded state; however, it offers the functionality to be collapsed by clicking an arrow located on the right side adjacent to the date.

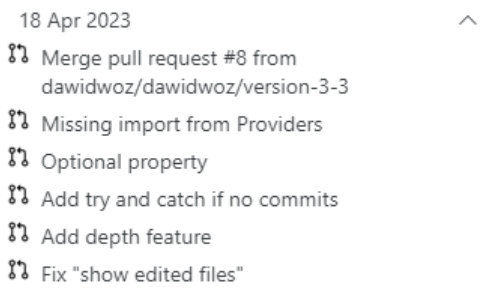

Fig. 2. An example single group of commits

1) Commit representation in General tab: One place to see the commit history is the *General* tab which is a preview view intended to facilitate easy access to the most recent alterations made to the object. We display the three latest days featuring commits. The rest of the commit list can be seen after clicking an option *+x more* where *x* is a number of currently not visible days.

2) Commit representation in the designed tab: Another way to access information about changes to the project is the *Commits* tab. Figure 3 exemplifies the user experience upon expanding and accessing this particular view.

3) Sorting options in Commits tab: On the top, we have the object title. It is in the same place as in the *General* tab. Below we have two sub-tabs for two groups of interaction - *Sorting* and *Filters*. The *Sorting* tab is active by default.

We have two sorting options by date or by author. They can be presented in ascending and descending order. We display 10 entries (dates or authors). The rest of the list can be expanded in the same way as it is in the *General* tab. When applying ascending order to the sorting of authors in the commit history, the listing is not based on alphabetical order, but rather on the order of their appearance within the commit history. This order of appearance is also maintained when considering commits associated with a particular individual, further adhering to the aforementioned rule. Figure 4 shows an example of the commit list sorted by author.

4) Filtering options in Commits tab: Another sub-tab is called *Filters*. Figure 5 serves as an illustrative depiction of the user experience upon accessing and opening this specific tab. The list can be filtered by the following factors starting from the top of the illustration:

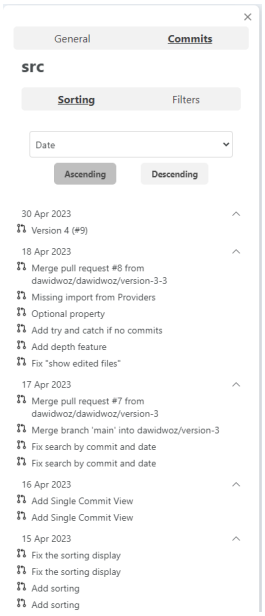

Fig. 3. The commit tab with the default view

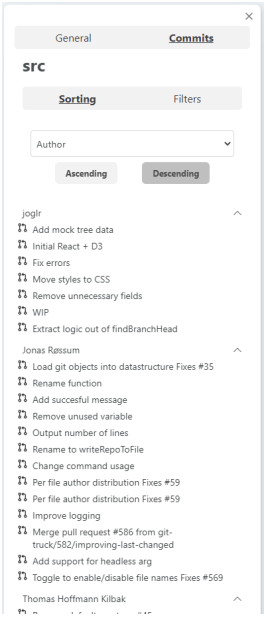

Fig. 4. The commit tab with the commit list sorted by author

- **A part of commit message** - For users seeking to search for a specific commit, a convenient method is provided whereby they can input a keyword expected to be present in the commit message. It is not case-sensitive. The whole phrase needs to be included in the commit message. RegEx is not supported. Multiple keywords are not supported.
- **A part of commit message or commit description** - In the event that a search by a partial commit message does not yield the desired results, users have the option to tick a checkbox provided below, thereby expanding the

search scope to include the commit descriptions as well. The same rule and criteria that apply to searching within commit messages remain applicable in this extended search functionality. By enabling this option, users can broaden their search parameters and potentially locate the desired commits by exploring the commit descriptions in addition to the commit messages.

- **Including or excluding authors** - A user might include or exclude a specific set of authors. Users can select authors from a dropdown list, which comprises all the authors who have contributed to the repository. This feature enables users to filter commits based on the desired set of authors, refining the displayed results to match their specific criteria and facilitating focused analysis or review.
- **Setting specific timeframe** -To afford users with further control and granularity, the system enables the specification of a timeframe in which changes were committed, allowing for precision down to half-hour intervals. Users have the flexibility to define the desired timeframe, thereby refining the displayed commits to align with their temporal criteria. If no specific timeframe is set, the displayed dates serve an informative function, dynamically showcasing the latest and oldest commit dates based on the current filters applied. This functionality empowers users to navigate and analyze commits based on temporal considerations, enhancing their ability to track and evaluate changes within the repository.
- **Including or excluding merge commits** - If a commit history includes merge commits incorporating phrases *"merge pull request"* or *"merge branch"* (default phrases for merge commits in GitHub), they will disappear while unticking a checkbox. This feature allows users to focus on commits other than merge commits.

5) Single commit view: When a user finds the desired commit, then they can click on the commit message to see more details about it. Figure 6 shows an example of the single commit view.

The provided information includes:

- Commit message
- Commit description
- Full commit hash
- Creation date
- Author
- An option to show edited files

To uphold security standards, the system does not display external objects, such as images and links, included within the commit description. This approach mitigates potential security risks associated with displaying or executing external content. Additionally, markdown formatting is not supported in the commit description, ensuring consistent rendering and eliminating any vulnerabilities that may arise from executing embedded markdown code. To enable a visual representation of the changes, GitTruck offers users the option to visualize the modifications made within a commit. GitTruck fetches

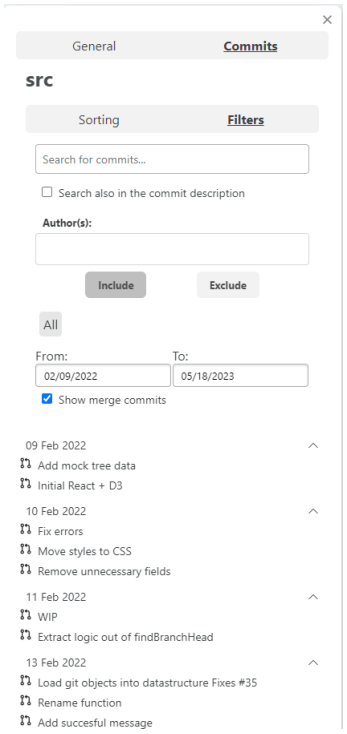

Fig. 5. The commit tab with the *Filters* sub-tab open

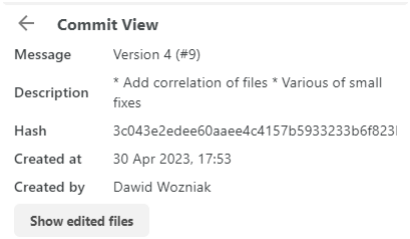

Fig. 6. The example of the single commit view

which files were changed with this commit and highlight the corresponding representation of those files in the visualisation. Additionally, all the files associated with the commit are listed below the visualization for easy reference. It is important to know that a file still needs to be present in the project to be highlighted and listed. Figure 7 shows an example of highlighting commit changes. Upon returning to the commit list, GitTruck reverts the visualisation back to its original state.

B. User evaluation

1) Multi-Stage evaluation: Our testers fulfil three roles: software engineer [P1] [P3] [P4] [P5] [P6] [P9], technical engineer manager [P7] and product manager [P2] [P8].

They have various range of professional experience with IT projects: three years or less [P5], between three and six years [P3] [P4] [P7] [P8], between six and ten years [P1] [P6] and more than ten years [P2] [P9]. Figure 8 provides detail visual representation of this data.

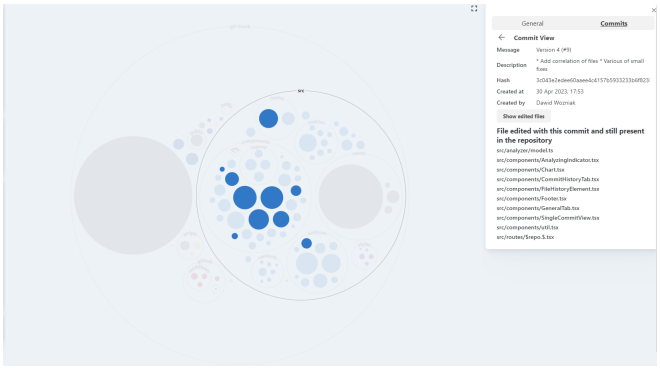

Fig. 7. The example of highlighting changes in the commit

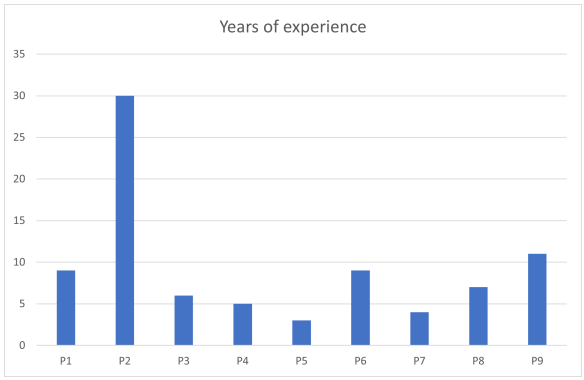

Fig. 8. The years of experience with IT projects of our multi-stage testers

Their average team size was defined in the range from three [P8] to twelve [P6]. Figure 9 provides precious data from our testers.

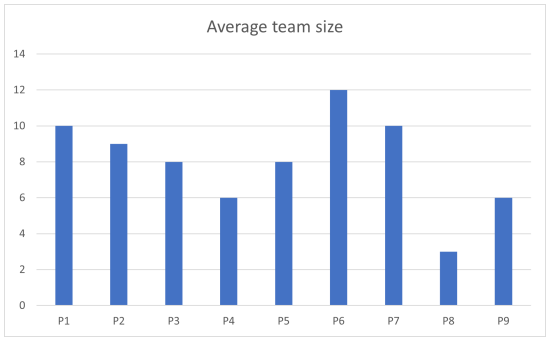

Fig. 9. The average team size of our multi-stage testers

In terms of the number of commits in the average project, it was answered very differently. Provided answers in a year perspective was roughly 365000 [P2], 36500 [P5] 3650 [P3], 1825 [P4] and 300 [P1]. The rest of the testers [P6] [P7] [P8] [P9] do not know the answer.

The average number of files in a project is usually not known [P1] [P2] [P5] [P7] [P8]. Project up to 1000 files [P4] [P6] [P9] and bigger [P3] were provided as average project size by our testers.

During the initial iteration with testers, we sought their opinions on the importance of commit messages. The results, as illustrated in Figure 10, unequivocally confirm the significant value placed on commit messages by the testers. An important consensus emerged among all the testers, highlighting the significance of maintaining a high-quality standard for commit messages.

Fig. 10. The importance of commit messages for our multi-stage testers

When we inquired about the level of improvement in our product since its initial version was presented to our testers, we were pleased to observe that they highly appreciated the changes made. The positive feedback received from the testers resulted in consistently high grades for this specific question. Figure 11 shows precious grades from our examiners.

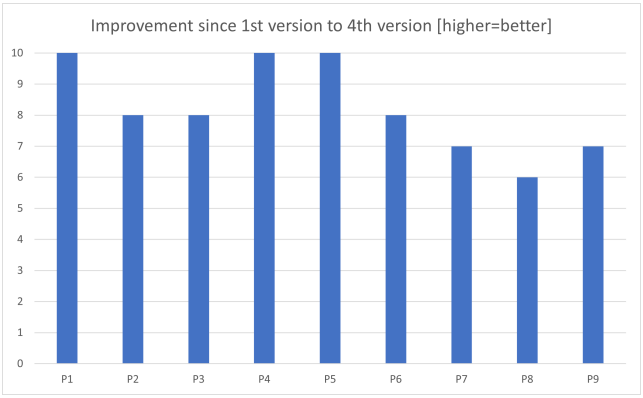

Fig. 11. The grade of improvement over four iterations from multi-stage testers

In order to assess the satisfaction of our testers with the UI representation of commit messages, we collected feedback during both the first and last iterations of our evaluation process. Figure 12 provides a visual representation of the distribution of grades assigned by the testers. Notably, it is evident that all grades have improved over time, indicating an overall positive shift in the perception of the UI representation of commit messages. Some of our testers [P2] [P4] [P6] appreciated a clear structure from iteration one. Nevertheless, the most improvement over time was requested and fulfilled in presenting commit details and integration with

visualisation [P2] [P3] [P4] [P6] [P9], indicating commits' author visually [P4] [P6] [P8], improving navigation among commits [P1] [P2] [P3] [P7] [P8] [P9], and making UI parts more flexible [P5]. During the final version development, several significant ideas were proposed by testers for further improvement, although they were not implemented. These suggestions included addressing accessibility issues [P7] [P8] [P9], making filters and sorting options collapsible [P1] [P8], representing more than one project branch in the visualisation [P6], supporting markdown syntax in commit description [P5] [P7], and displaying changes to the files [P9]. However, it is important to note that some testers did not identify any specific needs for visual improvements in the UI [P2] [P3] [P4].

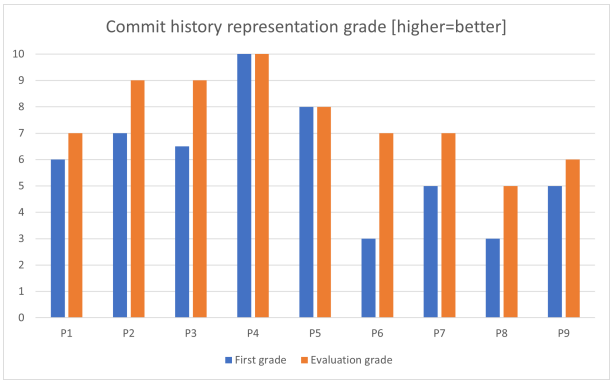

Fig. 12. The comparison between grades for the UI commit history representation from multi-stage testers

In addition to assessing the UI representation of commit messages, we also sought feedback from our testers regarding the overall usefulness of GitTrack. The evaluation results indicate a positive trend among our testers, with an increasing perception of usefulness over time. However, it is important to note a few noteworthy remarks. One tester [P9] did not provide an answer during the final iteration, citing the inability to estimate usefulness at that time. Another tester [P3] seems to have used a different grading scale, stating that the initial version of GitTrack would now receive a grade of one instead of the previously assigned six and a half. While not explicitly mentioned, it is possible that a similar case applies to testers [P2] and [P6]. Figure 13 shows the comparison between grades for our testers.

During the first interaction with our testers, their primary use cases for GitTruck were identified. These included browsing projects to gain insights [P2] [P3] [P6], exploring the codebase [P1] [P3] [P4] [P7], identifying high-traffic areas in the project [P2] [P4], reviewing changes to identify introduced bugs [P6] [P9] and and finding authorship information [P1] [P8]. These initial use cases remained consistent throughout subsequent interactions. Furthermore, some testers specifically highlighted the tool's value in exploring the codebase [P5] [P6]. Another group of testers acknowledged that GitTruck may not be directly suited for their specific roles, but they recognized its potential and utility [P2] [P4] [P8].

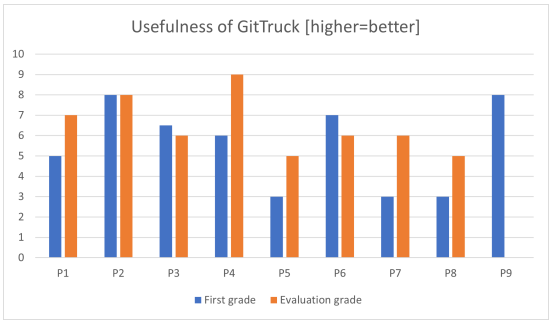

Fig. 13. The comparison between grades for the GitTruck usefulness from multi-stage testers

During our evaluation, we also asked our testers about the primary usage of the commit view in GitTruck. The responses varied among the testers, with each emphasizing different aspects and purposes. Some testers mentioned that they primarily use the commit view to find code ownership and identify the right person to contact [P1] [P3] [P8]. Others stated that their main usage is for general browsing of commits [P1] [P2] [P9], while some use it specifically to find a particular commit [P4] [P5] [P6] [P7]. Additionally, a few testers highlighted the importance of using the commit view to review changes and identify introduced bugs [P1] [P9], while others mentioned using it to find their own commits [P6].

As this information is essential for our research question. We list here some direct quotes from the transcriptions approved by our testers. These quotes highlight the perception of finding code ownership as a primary usage of the commit view:

- "The other one *[usage]* is like I have stuck on some bug. I'm working on some feature that I don't normally work on, and I want to know who to ask." [P1]
- "*[My primary usage is]* finding right person to contact" [P3]
- "*[My primary usage is]* finding, the right person who changed something and then I can ask why they change that" [P8]

Here are the quotes suggesting general browsing of commits as a primary usage of commit view:

- "Then, the last thing *[about usage]* is like I'm making a change and I wanna see like if I change this, what else should I change? So, I'm in some file and I know that if I change this, I need to change something else. So, I want to be able to go to this tool, find that file and then see a list of files, that were also changed and also see the commits that changed those, right?" [P1]
- "If I would have to look at commits, this tool will make it easier" [P2]
- "It will be mostly browsing comments, right? So, my main reason when I always look into commits is when I don't know what regressed something? Yeah. And that's my primary scenario for me." [P9]

These quotes reflect the viewpoint of our testers, indicating

that finding a specific commit is perceived as a primary usage of the commit view:

- ”I would go there if I wanted to find some specific commit. Specifically, if I was looking for a commit by commit message” [P4]
- ”I would still go use it for searching for specific commits.” [P5]
- ”Sometimes you work on a feature that is extremely complicated and then maybe you already made a pull request a long time ago that changed the specific file. So, you go back to that commit now you have to change that feature that you committed earlier, and you don’t remember where everything is hosted. So, you could go back to that commit, and you can see exactly which directories are all of these files being changed then.” [P6]
- ”I would probably just go there to find some *[commits with]* version - the new version or the old version of the software and then check, what was changed, maybe who did what, which parts change” [P7]

When asked about what is still missing in the project, several testers provided their suggestions. These included the integration of artificial intelligence [P6], further improvement in user experience [P1] [P8] and integration with their local Integrated Development Environment (IDE) [P4] [P7] [P9]. However, it is essential to note that some testers did not have any specific major ideas for additional features or improvements [P2] [P3] [P5].

2) Final evaluation: Our evaluation testers fulfil various roles: educator [P19] [P21] [P22], software engineer [P11] [P12] [P13] [P14] [P16] [P17] [P20], student [P10] [P15] and team manger [P14] [P18].

Figure 14 shows the years of experience with IT projects among our evaluation testers.

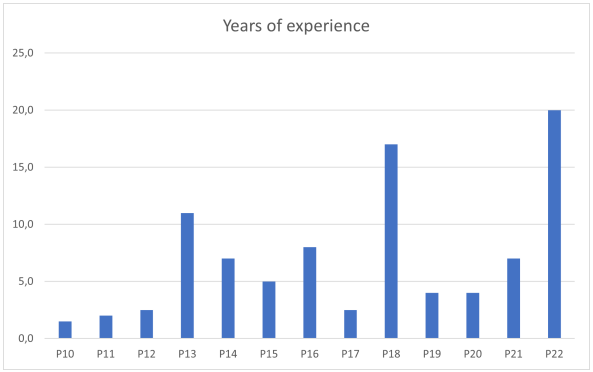

Fig. 14. The years of experience with IT projects of our evaluation testers

Figure 15 shows the average team size among our evaluation testers.

During the evaluation process, some of our testers shared valuable insights regarding the average number of files and the average number of commits in their projects on a yearly basis. This information provides a valuable perspective on the commit activity within their respective projects, helping us

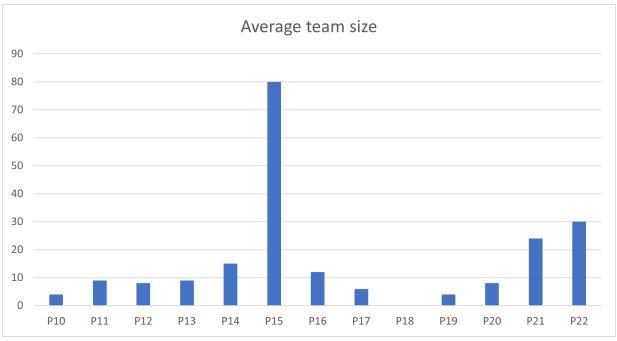

Fig. 15. The average team size of our evaluation testers

understand the scale and frequency of code changes made over time. Provided average number of commits are 36500 [P22], 21900 [P20], 11315 [P17], 1825 [P14], and 1095 [P12]. Provided average number of files are 600000 [P18], 100000 [P11], 1000 [P13], 800 [P20], 100 [P14], 71 [P19], 50 [P10], and 30 [P17].

Figure 16 shows the spread of grades for the user interface of commit history representation. Our design was appreciated with the lower grade being six [P15] through seven [P12] [P13] [P14] [P16] [P22] and eight [P10] [P17] [P20] to the highest rank nine [P11] [P18] [P19] [P21]. Users appreciate commonly UI being visually pleasant [P11] [P16] [P18] [P19], good structure [P19] [P22], profitable highlighting changes with a single commit [P14] [P17] and easy navigation [P10] [P15] [P20] [P21]. Testers complain about UI being too cluttered [P10] [P16] [P20], being too static [P16] [P21] [P22], being confusing [P14], being not clear while highlighting changes with a single commit [P15] and not providing visual introduction/suggestions to a user [P13] [P14] [P20] [P22].

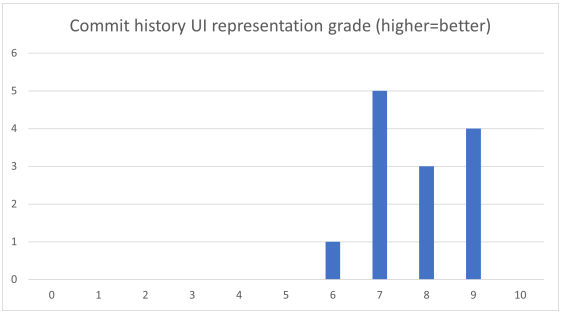

Fig. 16. The grades of evaluation testers for the UI commit history representation

Figure 17 shows the spread of grades for the usefulness of GitTruck with our changes. The grades vary starting from three [P12], four [P11], the most common five [P10] [P16] [P19] [P20] [P21] going through all numbers up: six [P15], seven [P13] [P22], eight [P17], nine [P14] and ten [P18]. People complain commonly about it being not suitable for their role [P10] [P12] [P20], being slow and not scalable well [P12] [P16] [P18], navigation to the object not embedded in URL [P12], UI not providing a good overview [P13] [P15],

not being integrated with their Git-solution provider [P16] [P20] and not bringing values they care about [P11] [P19] [P20]. Users appreciate easy way to define code ownership [P10] [P12] [P17] [P18] [P20], helping with their job tasks or potential job tasks [P10] [P18] [P21] and easy access to the visualisation and commit information [P19] [P22].

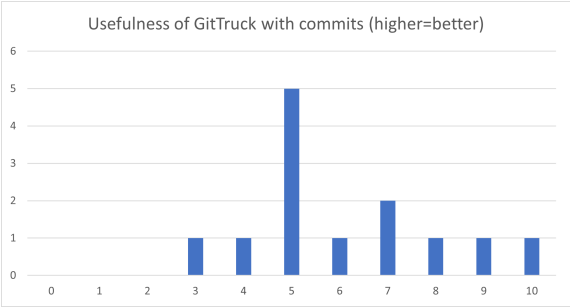

Fig. 17. The grades of evaluation testers for the GitTruck usefulness with our changes

In our evaluation, the question of the primary usage of our commit view was raised. The most common use cases, mentioned by our testers, are: searching for general commit [P11] [P15], or more specifically the commit that introduced and/or fix some bug [P12] [P18], finding the code ownership [P13] [P17] [P18] [P20] [P21], finding their own contribution [P10], assessing the difficulty of future tasks [P13] [P18] [P22], checking and grading students participating in the project [P19] [P21] and finding places with the huge traffic in the project [P13] [P16] [P18].

As this information is essential for our research question. We list here some direct quotes from the transcriptions approved by our testers. These quotes highlight the perspective of our testers, confirming that the main usage of the commit view is focused on searching for commits, especially looking for commits that introduced/resolve bugs:

- ”Looking for changes like in a ”Broadway” where I’m not sure what was the commit and what files were changed. That’s basically my first use case.” [P11]
- ”I have a bug and I have to go a few commits back to see what has been changed like exact lines basically.” [P12]
- ”This information *[about commits]* I would say this is the most important. This is what the user should focus on in your project. The filtering is less important” [P15]
- ”Ideally, I have a filter and then it shows some kind of relations between some bubbles in some circles where changes happened but in some there was not the change. It will be super great to see because then I could clearly see where the change was not happening. And it’s very, very interesting, you know, in terms of errors, which is in maintenance mode, errors which might happen, you know, it is bad coded.” [P18]

Here are the quotes suggesting finding ownership as a primary usage of commit view:

- ”Usually in my day-to-day I will be looking at the others commits or blame of a specific file to the get an idea for who might have touched this last, because what you care about is the very line that you suspect of being broken” [P13]
- ”It’s very nice that you can set multiple authors and then see on which files they have like collaborated instead of the very binary view that we have usual where it’s just one person getting all of the credit. I think it’s very nice and you can put in two or three names and see where they overlap.” [P17]
- ”I would use this to try to understand who does what.” [P18]
- ”I don’t know if I’d be interested in what one of my coworkers has done in the last year. Maybe my manager would be interested in seeing that, but maybe we have like some service accounts that you know autocreates PR and stuff that maybe sometimes that could be useful *[to spot]*.” [P20]
- ”Such kind of tool is really interesting when we are many collaborators on which you can see here in which part each collaborator works” [P21]

One tester suggested finding their own contribution is a primary usage. Here is a direct quote to confirm it:

- ”If I’m working on a project and I remember at some point I changed something *[...]* I can actually go and use this tool to figure out because a lot of times you have done something previously that you want to go back to, right?” [P10]

According to the feedback from two testers, they have suggested that the tool could be utilized for grading students. Here are the quotes confirming this perspective:

- ”I’m just gonna look at all the refactorings now and see - Can I spot like learning outcomes?” [P19]
- ”Another use case where I can see really interesting your work when we need to evaluate a work of students for a project” [P21]

The commit view is also valued for its ability to identify areas with significant traffic and assess the difficulty of future tasks within the project. Here are quotes from testers confirming this primary use case:

- ”If you’re a person task with some sort of team organisation, so when you are like team lead manager or whatever level. You might be interested in to see it. If you have a project spanning multiple teams, there is a nice way of breaking the responsibility to multiple smaller teams” [P13]
- ”It would be great for understanding that maybe there is some specific area that has huge traffic, right?” [P16]
- ”Sometimes we have also like management level discussion who is really specialised and how much they really did versus someone else and it’s for the quantity of data. So in period of every three months you’ll verify all files which never changed.” [P18]
- ”It’s sometimes when ones need to quickly assess how difficult something could be addressed as a task” [P22]

In the evaluation process, there were discussions regarding a missing feature, particularly in relation to the commit message history. The proposed ideas include incorporating a timeline feature into the visualization, allowing users to observe changes over a specific time period [P18] [P19] [P22]. Another suggestion involves automatically evaluating the quality of commit messages [P18]. Additionally, recommendations were made to improve the user interface, including adding visual tips and refining the overall UI experience [P12] [P13] [P16] [P22]. Adding support for AI tools [P11] and adding a link to Git-solution provider [P16] [P20] were also mentioned. Some people did not have any major ideas of the features they also would like to have here [P10] [P14] [P15] [P17] [P21]. These suggestions and feedback provide valuable insights for potential future enhancements to the system, ensuring its continuous improvement and alignment with user needs and expectations. Many of them were not implemented only because of the time limitation and can be seen in the future work - Section VI.

C. Ideas that were not desired

1) Classification of commit messages: In the past, commit messages were seen as a text classification problem. This approach was mentioned by some testers. [P2] [P4] [P6] [P11] [P19] However, they would prefer to use tools powered by artificial intelligence. The testers express a desire to leverage the capabilities of AI-driven solutions, which can offer more advanced and sophisticated techniques for analyzing and processing commit messages stating the naive keyword method would not provide them with a good level and accuracy of the results.

2) Tag cloud based on commit messages: In the second feedback iteration, we specifically inquired about the concept of implementing a tag cloud based on commit messages to our multi-stage testers. This idea was generally perceived as lacking usefulness [P3], being viewed as promotional material [P8], or not deemed profitable for their respective roles [P4] [P7]. A minority of testers expressed the belief that it could potentially be useful, albeit not in their particular role, without providing specific use cases [P2] [P6] [P9]. One tester [P1] suggested using filenames instead of commit messages to see which files frequently changed with the given set of filters.

3) Frequently changed files - Correlation matrix: In the third feedback iteration, we asked our multi-stage testers about the idea of listing frequently changed files based on commits. It was supposed to create a correlation matrix of files and potentially discover non-obvious dependencies between files. This idea encountered resistance due to concerns over its perceived complexity [P3], the notion that it was unnecessary [P7] [P8] [P9] or the suggestion that it should be implemented as a separate tool [P1]. Some testers acknowledged the potential usefulness of the idea [P4] [P5] [P6], but without specifying precise use cases and noting that it may not be applicable to their particular roles [P2].

4) Indication of commit size: In the third feedback iteration, we asked our multi-stage testers about the idea of indicating commit size. The reception of this idea varied among testers. Some regarded it as a favorable idea [P3] [P4] [P5] [P6]. Some said it might be useful, but not for their role, not preciously defining use cases [P2]. Two testers [P1] [P7] suggested that analyzing the impact of changes to the repository, rather than focusing on statistical changes, would be more valuable. One tester [P9] does not think it is needed. Additionally, it was also not clear if commit size shall be measured in the number of touched files or changed lines.

IV. Discussion

A. Limitations of the sampling

The presented user evolution was performed with a sample of 22 participants which was enough to define the desired role of commit messages and confirm the usefulness of our implementation. However, a larger study could have been made. Such a study would require the selection of desired roles and experience, so the study will not be biased. It was not done due to the time limitation of this project.

It is worth noting that the varying levels of familiarity with GitTruck among testers, although potentially considered a threat to validity, were not biased as the implementation of the new feature did not heavily rely on prior experience with the tool.

Another approach to gathering more data would require some telemetry and a built-in survey in GitTruck. It was not done as we did not wish to breach the initial GitTruck rule of not sending any data externally.

B. Limitations of the meeting formula

Our methodology assumes meeting with testers in formula one-to-one. Despite its obvious advantages, such as an opportunity to adjust the presentation to the tester's role and technical awareness, avoiding the risk of being biased by other people or last-minute cancellation due to other responsibilities, it has also disadvantages. This method is heavily time-consuming which limited the number of participants in the user evaluation. Some people might also feel anxious and stressed during a meeting, resulting in not expressing all opinions, especially negative comments.

The potential for unwanted misinterpretation exists, as similar opinions expressed by testers may have different underlying meanings or nuances that require careful consideration. The ambiguity regarding whether GitTruck should be run on the tester's machine or our machine introduced a mixed approach, potentially introducing bias factors that need to be taken into account during the evaluation.

C. Limitations of GitTruck

The current version of GitTruck has a problem with the scalability of big projects. More details can be found in Section VI-A. As a temporary measure, we limited GitTruck to take only 5000 latest commits in order to make it analyze desired repositories by our testers [P1] [P2] [P3] [P5] [P6] [P8] [P9] [P11] [P12] [P13] [P16] [P18] [P20]. This limitation can potentially be seen as a potential bias factor but we believed

it would have been worse if we had not shown the desired repositories to our testers.

D. Commit conventions

We didn't record if a tester uses any commit conventions in their repositories. This information can be partly detected based on answers to other questions but it seems it might be an important factor in how the product is received. Having a commit convention forced in the repository commit history might influence what kind of interactions a user would like to have. We would like to create a product that is friendly for convention and non-convention commit history, so there might be potentially overseen bias in our user evaluation. It can be seen especially bright for one tester [P7] who emphasizes the role of commit convention in terms of our implementation usefulness.

V. Related work

In addition to gathering feedback from our testers, we conducted research to explore the current state of knowledge and existing tools related to commit message interaction and representation. This research served as a valuable source of inspiration during the development of GitTruck.

The research highlighted a clear need for a hierarchy-oriented visualization tool that effectively incorporates commit messages with a proven role in navigation among them [14]. Johannes Feiner and Keith Andrew [11] presented a paper about the hierarchy-oriented tool RepoVis where the commit messages are represented along with other commit metadata. This work in RepoVis was an inspiration for our single commit view - Section III-A5.

Commit messages might be also used to perform the analysis in order to suggest refactoring. [5] They created a new approach called RefCom to perform such an analysis using a keyword-based method. A similar approach was implemented in other papers [6] [7] where they classify commits into three categories using Natural Language Processing (NLP). It could have been potentially an interaction with commit messages inside GitTruck but our testers would prefer to use a more sophisticated approach using AI-powered tools showing new expectations of developers powered by technology advancement.

Potential desired interactions and UI were designed similarly to the tool called eazyBI [9] where the authors presented a more visual statistical analysis of commit which was the inspiration for our visualisation of a single commit in GitTruck.

Potential integration with Git-solution provider mentioned in the future work - Section VI-C - is also supported by the part conclusion of the research [13] on commit message quality where they stated, it is needed to consider both commits messages and link contents while estimating a commit message's quality that might be a use case for some of our testers [P18].

VI. Future work

A. Scalability

The scalability issue was mentioned as a significant show-stopper for our testers. More details can be found in Section IV-C. The main problem with the current infrastructure is its strategy to preload everything that is saved to a JSON file. This file is loaded into the memory so if it is no big to be loaded, a browser crashes. Solving this issue can increase the overall user experience while working on huge projects. However, it is particularly complex in the current infrastructure providing a user with a tool using executable npm package binaries (NPX).

B. Integration with AI tools

In Section III-C1, we described the rejected idea to classify the commit. However, some testers [P2] [P4] [P6] [P11] [P18] pointed out it would be good to integrate the product with AI tools, for example, ChatGPT. This capability has the potential to be leveraged for commit message analysis and the generation of a comprehensive summary of changes associated with the provided set of commits.

C. Integration with GIT-solution provider

The current requirement of installing the tool externally is recognized as a potential accessibility barrier by our testers. It might be fixed by integration with a GIT solution provider, for example, GitHub, GitLab or Azure DevOps, which was suggested by our testers. [P1] [P4] [P7] [P9] [P11] Implementing such integration would enhance the usability and accessibility of the tool, thereby improving its adoption and effectiveness among users. Such integration might also have a positive impact on scalability and willingness to use our commit messages representation and integration over the default view provided by the solution provider.

D. Tweaking UI parts

During our evaluation, a few user interface suggestions were made. While we successfully implemented some of these suggestions, it is worth noting that more complex requests were deferred as future work due to their intricacy or resource requirements. The most important suggestions include most visible branding [P2] [P7] [P8], more flexibility in customizing UI [P1] [P3] [P5] [P22], a new type of visualisation [P2] [P8] [P15] [P22], UI introduction tour [P13], dynamical grouping dates, for example, last week, last month, last year, based on a number of commits [P14], excluding commits including certain keywords [P3] [P5], improving tool accessibility [P3] [P9] [P22] and more smaller improvements. Considering that suggestions made during the evaluation process can sometimes be subjective and influenced by individual taste, seeking UX consultation can be highly beneficial for improving the overall user experience, including the commit history view.

E. Select the target group

Looking through the feedback, it is possible to spot that different roles require different features. Their perspective on the tool, including commit messages representations and interactions, is different. Therefore, we believe it would be beneficial for future development to define the target group or create specific sub-versions for various roles, for example, lecturer, software engineer, product manager, team manager

etc. Then, we can tailor UI and functionality so they fit the best scenarios typical for the selected role.

VII. Conclusions

In conclusion, we would like to answer briefly our research question. We believe the representations of commit message history as the list in an ordered manner with sorting and filtering options are beneficial in the hierarchy-oriented visualization. Desired interactions involve the detailed commit view and visualization of the commit changes. However, it is important to note that the perception of these features and their usefulness varies based on the tester's role. Therefore, considering the specific needs and perspectives of different user roles is crucial for optimizing the commit history view and providing a tailored user experience.

VIII. Appendix list

The appendix list contains all transcriptions of meetings that we have done in order to collect feedback. All transcriptions have been approved by the interviewed person. They can be found online under the link: github.com/dawidwoz/thesis-appendix. We include also the Excel spreadsheet used to create diagrams and Fig.1, Fig.3, Fig.4, Fig.5 and Fig.7 in full resolution.

[P1] Transcriptions of four meetings with the tester - Ayrton
[P2] Transcriptions of four meetings with the tester - Błażej
[P3] Transcriptions of four meetings with the tester - Enrico
[P4] Transcriptions of four meetings with the tester - Jakub
[P5] Transcriptions of four meetings with the tester - Julia
[P6] Transcriptions of four meetings with the tester - Matti
[P7] Transcriptions of four meetings with the tester - Michał
[P8] Transcriptions of four meetings with the tester - Paulina
[P9] Transcriptions of four meetings with the tester - Monica
[P10] Transcription of one meeting with the tester - Sofie
[P11] Transcription of one meeting with the tester - Balàzs
[P12] Transcription of one meeting with the tester - Denis
[P13] Transcription of one meeting with the tester - Daniel
[P14] Transcription of one meeting with the tester - Jonas
[P15] Transcription of one meeting with the tester - Kristoffer
[P16] Transcription of one meeting with the tester - Marta
[P17] Transcription of one meeting with the tester - Thomas
[P18] Transcription of one meeting with the tester - Ievgenii
[P19] Transcription of one meeting with the tester - Bjørn
[P20] Transcription of one meeting with the tester - Alexander
[P21] Transcription of one meeting with the tester - Benoît
[P22] Transcription of one meeting with the tester - Leonel

IX. References

[1] Stack Overflow Developer Survey 2022. URL: survey.stackoverflow.co/2022/section-version-control-version-control-systems accessed: 2023-05-22.
[2] Amel Hamidovic, Jakov Matic, Günther Kniewasser, Andreas Wöls "Software Repository Visualisation" Graz University of Technology A-8010 Graz, Austria, 16 April 2018
[3] GitTruck repository. URL: github.com/git-truck/git-truck
[4] K. Højelse, T. Kilbak, J. Røssum, E. Jäpelt, L. Merino and M. Lungu, "Git-Truck: Hierarchy-Oriented Visualization of Git Repository Evolution," 2022 Working Conference on Software Visualization (VISSOFT), Limassol, Cyprus, 2022, pp. 131-140, doi: 10.1109/VISSOFT55257.2022.00021.
[5] Rebai, Soumaya, Marouane Kessentini, Vahid Alizadeh, Oussama Ben Sghaier and Rick Kazman. "Recommending refactorings via commit message analysis." Inf. Softw. Technol. 126 (2020): 106332.
[6] M. U. Sarwar, S. Zafar, M. W. Mkaouer, G. S. Walia and M. Z. Malik, "Multi-label Classification of Commit Messages using Transfer Learning," 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 2020, pp. 37-42, doi: 10.1109/ISSREW51248.2020.00034.
[7] S. Gharbi, M. W. Mkaouer, I. Jenhani, and M. B. Messaoud, "On the classification of software change messages using multi-label active learning," in Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 1760–1767.
[8] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic classication of large changes into maintenance categories," in 2009 IEEE 17th International Conference on Program Comprehension. IEEE, 2009, pp. 30–39.
[9] EasyBI - Git Commit Log Analysis - URL: eazybi.com/integrations/git
[10] Long, Suzanna. (2010). The strategic implications of non-technical stakeholder acceptance in high technology system design and implementation. Human Systems Management. 29. 10.3233/HSM-2010-0716.
[11] Johannes Feiner and Keith Andrews. "RepoVis: Visual Overviews and Full-Text Search in Software Repositories". In: Sept. 2018, pp. 1–11. DOI: 10.1109/VISSOFT. 2018.00009.
[12] GitTruck Beta repository. URL: https://github.com/dawidwoz/git-truck
[13] Jiawei Li, Iftekhar Ahmed "Commit Message Matters: Investigating Impact and Evolution of Commit Message Quality" Donald Bren School of ICS, University of California, Irvine
[14] Pooja Rani "Assessing Comment Quality in Object-Oriented Languages" Inauguraldissertation der Philosophisch-naturwissenschaftlichen Fakultät der Universität Bern