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

K. Højelse*, T. Kilbak*, J. Røssum*, E. Jäpelt*, L. Merino†, M. Lungu*

*IT University of Copenhagen
†DILAB, Escuela de Diseño, Escuela de Ingeniería, Pontificia Universidad Católica de Chile

***Abstract*—The assessment of repository evolution is particularly complex when one needs to analyze both 1) how the codebase is organized hierarchically and 2) how this codebase evolves over time. To address this problem, we developed Git Truck, a visualization tool that includes multiple views for the analysis of the evolution of hierarchically organized Git repositories. We conducted a preliminary user evaluation with 18 participants who, using a remote and asynchronous method, installed Git Truck, used it, and filled in a questionnaire to report their experience and impressions. We learned that participants consider particularly useful views that help them with understanding the contribution level of team members and views that highlight the parts of the system that change the most. The participants see Git Truck as a highly specialized tool; not for daily use but rather for a lower frequency of use.**

I. Introduction

Most industrial software projects are large, complex, and impossible to create or maintain without a team of highly skilled developers [6]. One fundamental technology that such teams of developers employ to collaborate are *version control systems* – with Git being the most popular such system at the moment [5, 25]. Often, software projects' stakeholders would benefit from using insight from repository evolution in their decision making processes, however, exploring the structure and evolution of a software repository is still a challenge [3].

One of the most powerful conceptual tools for managing complexity is the hierarchical organization of the source code of such systems [14, 20, 27]. This is why many hierarchy-oriented visualization tools have been proposed over the years. However, many of these are language dependent and do not show evolution [4, 24, 26], many take evolution into account but their focus is not the hierarchical organization of the source code of the system [10, 12], and finally some are presented without an evaluation with users [11] as a recent survey also observes [19].

In this context we think that it is valuable to further investigate with users in which way a language-independent, hierarchy-oriented evolution visualization tool can support stakeholders in their software engineering practice. Consequently, we formulate the following research question: *How can hierarchical-oriented visualization be used to support the stakeholders of a project in assessing the evolution of the system?*

To address this question, we develop *Git Truck*, a tool that can visualize Git repositories for the analysis of evolution. Git Truck uses hierarchical visualizations (i.e., Treemaps, Circle packing) to represent files in a repository. The system shows files nested in folders to provide a spatial locality of a given codebase. In the visualizations, the marks that represent files are augmented with code metrics using size and color. The design of the system and visual perspectives are intentionally generic, such that, the results are potentially transferable to other tools. The usability of the tool is on the other hand, fairly good, such that, the tool itself does not stand in the way of potential users trying it out and reporting on it.

To evaluate Git Truck, we first invited personal contacts and conducted a pilot study. We adopted a remote asynchronous method in which participants were asked to *(i)* install Git Truck in their working environment, *(ii)* use it to visualize a Git repository, and *(iii)* report their experience by filling in a questionnaire. We used the results of the pilot study to improve the questionnaire by adding, removing and clarifying questions. Next, we announced an open invitation using social media (i.e., LinkedIn, Twitter), in which we engaged 18 participants. The anonymized results from the questionnaires are available as supplemental material [15].

We found that 86% of participants had no problems installing Git Truck, and 77% found the "Top Contributor" view useful. Overall, participants particularly were interested in identifying files developed by single authors as well as in the analysis of the contributions of team members. They think Git Truck could be specially useful for novice developers involved in large projects.

The main contribution of this paper is twofold: *(i)* presenting the publicly available Git Truck visualization tool and its architecture, and *(ii)* the results of a remote asynchronous user evaluation that sheds light on the usefulness of Git Truck and similar tools

II. Git-Truck

We present Git Truck - a tool to obtain a visual overview of a Git repository, by displaying a visualization in a web browser. The tool is run locally on the user's system, enabling the visualization of private repositories and repository host independence[1]. The link to the repository containing the solution can be found on Git Truck's GitHub page[2]. The tool can also be easily installed from the NPM registry[3] by running `npm install git-truck`.

[1]Can work with any Git provider, e.g. GitHub, Azure DevOps
[2]https://github.com/git-truck/git-truck
[3]https://www.npmjs.com/package/git-truck

A. Layouts for visualizing hierarchical file organization

To visualize the structural information of a repository, i.e. files and folders, we decided on two different algorithms: circle packing[4] and treemapping[5]. These algorithms are used to generate the diagrams bubble chart (Figure 1) and tree map (Figure 2), respectively. Both diagrams are nested hierarchical layouts, where folders are represented by outlined shapes, while files are represented as filled shapes. For both diagrams, the area of a file representation, is a function of the files size in bytes.

In general, the tree map leads to less empty space than circle packing. See Figure 1 and Figure 2 for a comparison of the utilized space between the two chart layouts (note that both show the same system: Git Truck itself). We chose to use both, to make the tool suited for a greater set of repositories sizes.

Folder names are shown on folders that are big enough to fit the name, while file names can be seen when hovering over its representation, with the cursor. This is done to reduce the clutter that displaying all the file names cause, in addition to removing performance issues related to rendering a lot of text.

Both layouts are further optimized, by *(i)* not rendering elements that would become too small to see, *(ii)* not rendering empty folders, and *(iii)* collapsing any folder that only contains a single folder, into a single element.

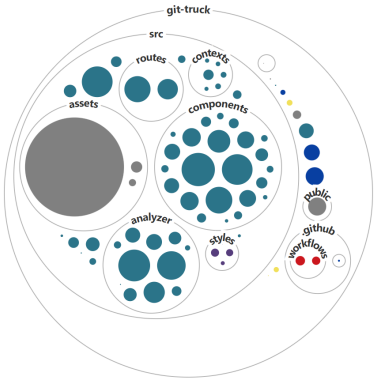

Figure 1. The Git Truck repository visualized with circle packing.

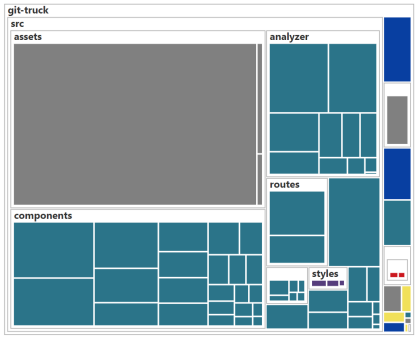

Figure 2. The Git Truck repository visualized with treemapping.

[4]https://en.wikipedia.org/wiki/Circle_packing
[5]https://en.wikipedia.org/wiki/Treemapping

B. Predefined Views on a Repository

Git Truck comes with a set of five predefined methods of highlighting various properties of a repository. They are enumerated and described in this section.

1. File Extension

This perspective gives an overview of where different types of files are located in a repository. A file is colored according to its extension, in accordance with the colors used on the GitHub website. This view is intended to strengthen the structural overview of a repository, by showing how files of different programming languages are organised. This is useful, since most of the systems nowadays are multi-lingual [22]. It also serves to help the user decide whether some folders or files should be removed from the analysis.

Figure 3, shows the File Extension view[6] applied on PyTorch – arguably, the most popular machine learning library at the time of writing this paper. The figure shows that different subsystems are written in either dominantly C++ (e.g. the `torch` module in the top-left part of the image) or Python (e.g. the `test` module in the top-right part of the image).

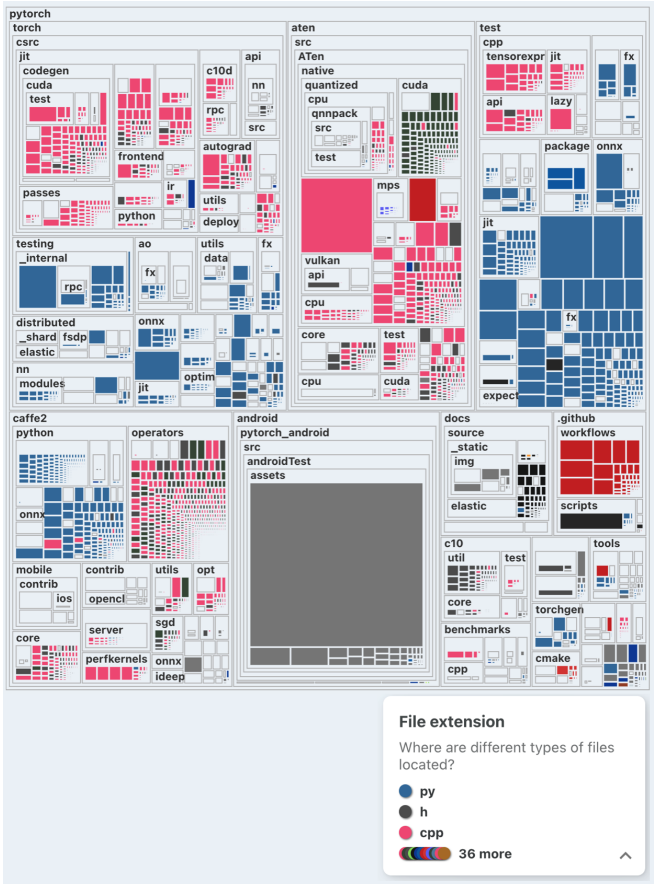

Figure 3. The File Extension View for PyTorch shows an implementation that is split between C++ (pink) and Python (blue)

[6]All the example views presented in this section are based on the full history of the subject systems as of June 2022. For PyTorch we used the code at: https://github.com/pytorch/pytorch

2. Number of Commits

This view highlights those files in the history of the repository, that have been affected by the largest amount of commits and is intended to help a user find "hotspots" in the code base, i.e. which files are changed frequently.

Figure 4 shows the Number of Commits View on the Flask web application framework. Flask[7] is one of the two most popular web application frameworks for Python [8]. What the figure confirms is that Flask deserves its self-designation as a *microframework* – the source code is quite small, examples, tests, and docs take most of the view. Also, the two most changed files are `src/flask/App.py` and `CHANGELOG.md`.

The number of commits for a file, is the count of commits reachable from the selected branch head, that also makes a change to that particular file. The range of commit counts are normalized, whereafter the color is picked from a linear color gradient. A low commit count corresponds to a high lightness value and vice versa.

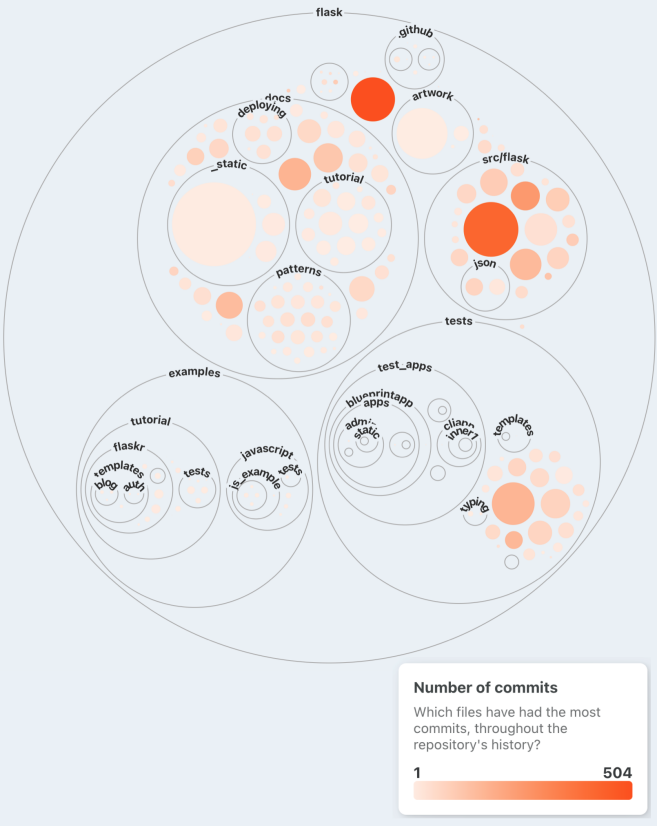

Figure 4. The most changed files in Flask web application framework are `flask/CHANGELOG.md` and `src/flask/App.py`

3. Last Changed

This view shows the time that has elapsed since the latest modification, of the various files in a project, occurred. The color is picked from a scale where a darker color indicates a higher value, i.e. that a file has not changed for a longer time, and a light color denotes a file that has been modified recently. The color scale is exponential with six fixed levels denoting the following periods: a day, a week, a month, a year, two years, more than 4 years.

Figure 5 presents the same Flask system as the previous view. It shows that the files in the `/src` and the `/tests` folders changed more recently than those in the `/docs` folder.

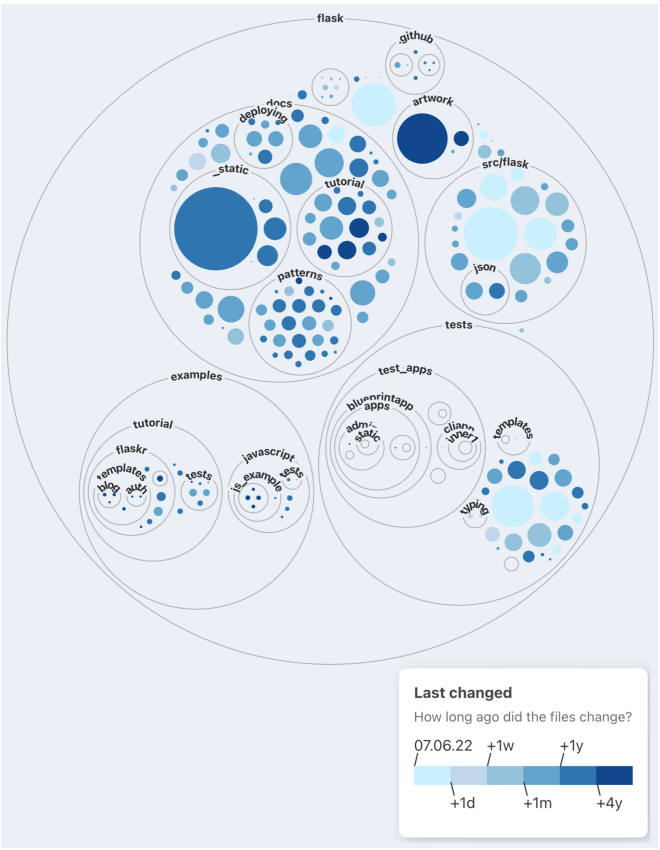

Figure 5. Screenshot of the Last Changed View for Flask shows a system in which code (`/src`) and tests are recently changed (as of 2022)

There are very few files that have not changed in the last four years. Compare this with the well-known open source project, ArgoUML represented in Figure 6.

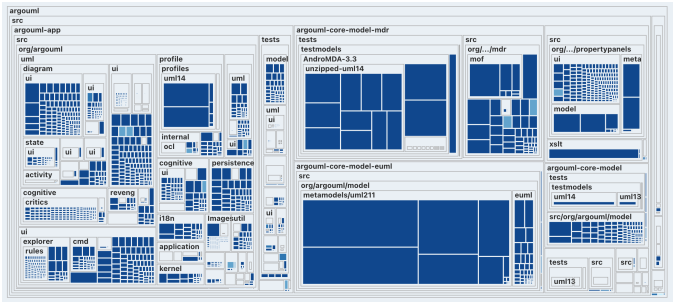

Figure 6. Last changed in ArgoUML shows a system in which most of the files did not change in the last four years

[7]https://github.com/pallets/flask

[8]It was the top most popular framework for Python and the 7th across all languages in the StackOverflow developer survey from 2021 (https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe)

4. Single Authors

This view is intended to highlight files which have a truck factor of one[9], indicating that only one person has a deep understanding of the file.

This view highlights in red files that throughout the repository's history are authored by only one person. Files that have more than one author are colored in a neutral color.

Figure 7 shows single authors in sqlalchemy[10] - an ORM for Python that is often used with Flask for backend development. The figure shows that very few files have only one single author in this system. This although the "Author distribution" sidebar shows that one author in the system is responsible with 89% of the code changes as measured in LOC[11].

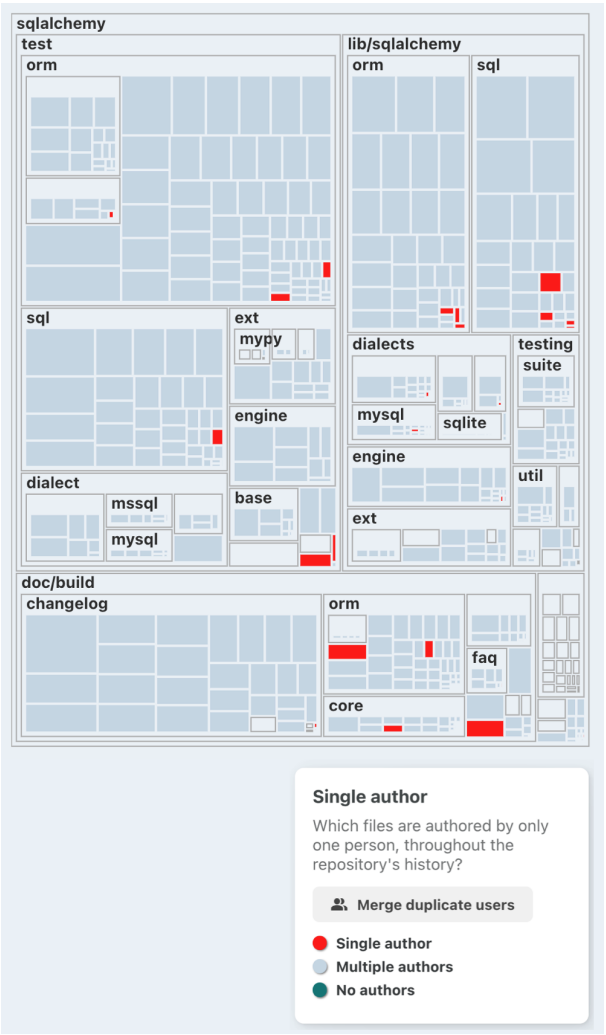

Figure 7. Screenshot of showing the Single Authors View for SQLAlchemy – the most popular ORM library for Python

5. Top Contributors

This view is intended to help a user get an overview of who is most familiar with parts of the codebase. It shows which author has made the most line-changes to a particular file[12] throughout the history of the repository. Author colors are generated using a method that aims to assign a visually distinct color to each author.

Figure 8 presents the top contributors for sqlalchemy – the system presented also in the previous section. The figure shows that virtually all the files are dominated by one developer.

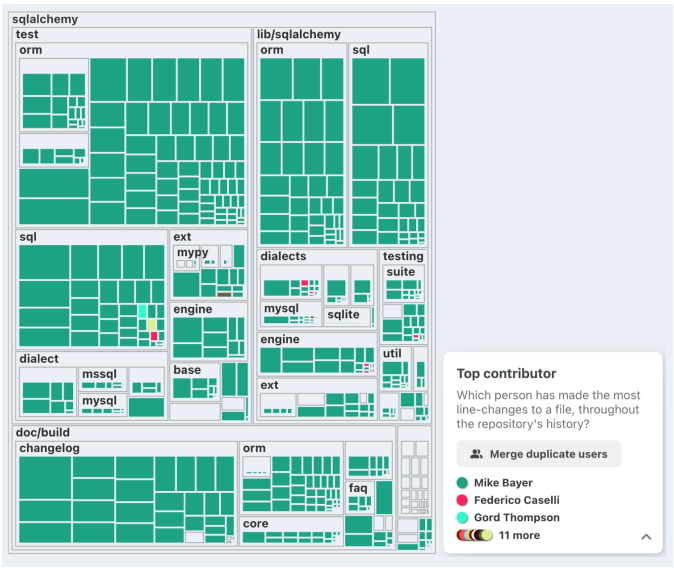

Figure 8. The Top Contributor view for SQLAlchemy highlights a critical open-source system where a single author practically dominates all the files.

Compare the figure above with the one from another popular ORM – EFCore[13] for the .NET platform. This one is also dominated by one contributor, but not to the same degree as the previous system.

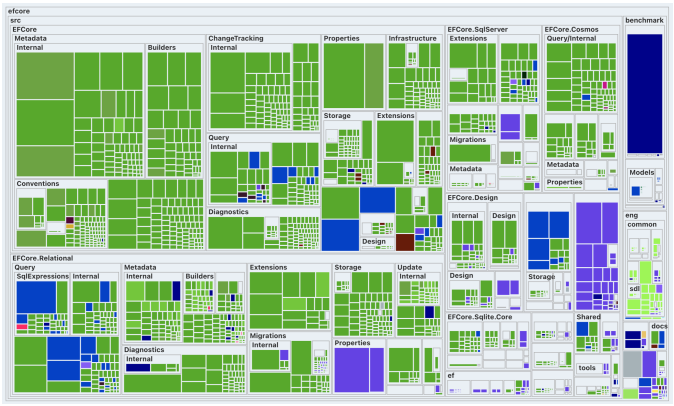

Figure 9. The Top Contributor View for EFCore – an ORM framework for the .NET platform

[9]https://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/
[10]https://github.com/sqlalchemy/sqlalchemy
[11]Lines of Code

[12]The most line changes are calculated in a first-past-the-post manner (https://en.wikipedia.org/wiki/First-past-the-post_voting)
[13]https://github.com/dotnet/efcore

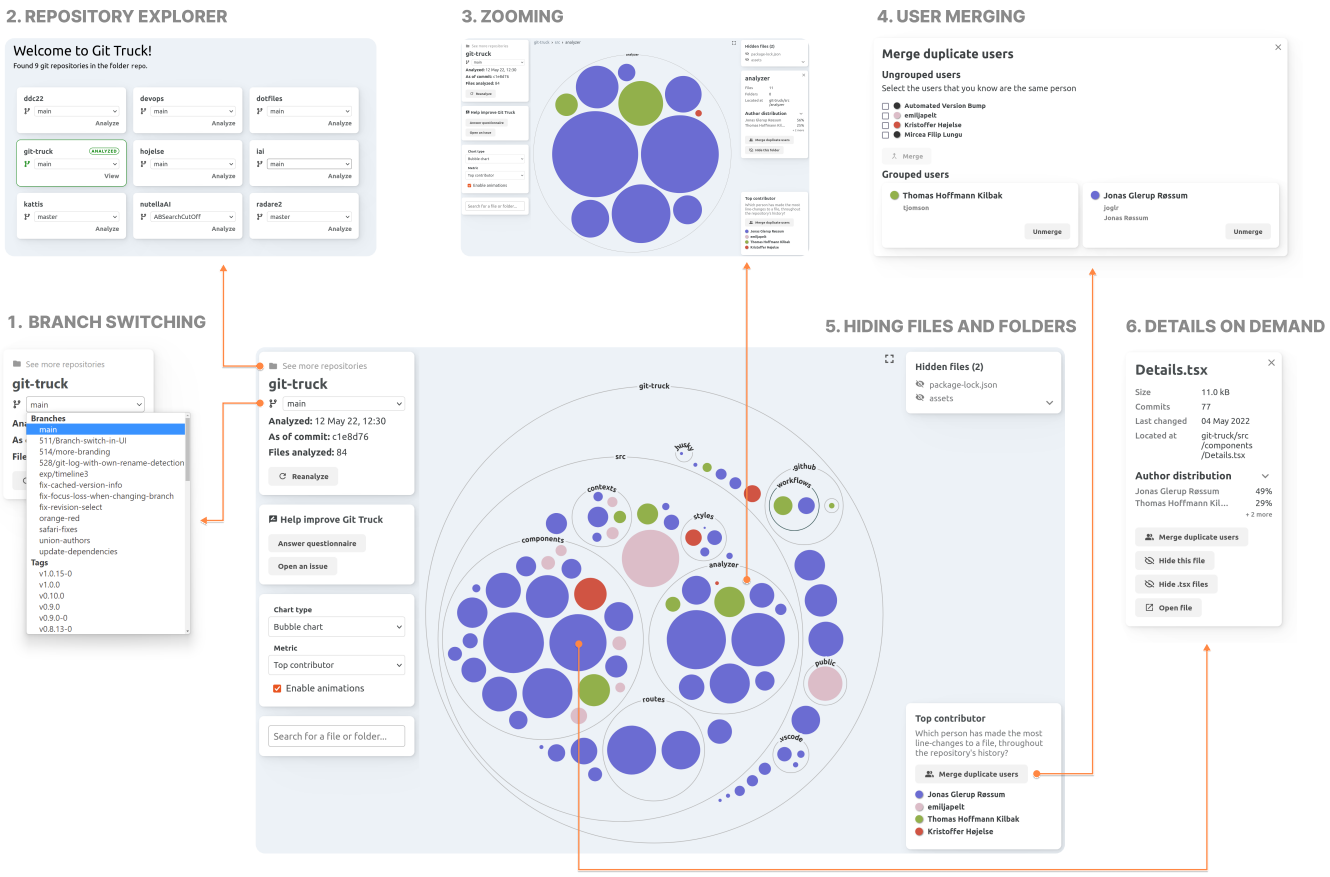

Figure 10. Overview of the Git Truck UI. The orange arrows point from a clickable target to the component or page that will be shown once clicked.

C. Interaction and Navigation

The following list of interactive features have been implemented in Git Truck to support exploring repositories.

1) Branch switching: Selecting a different Git branch or a different tag will analyze it and then transition to presenting the new information. This allows the user to see how the project has changed between multiple points in time.

2) Repository explorer: When opened in a folder that is the parent of a series of repository folders, Git Truck provides an overview of all the repositories in a given folder. However, this is a feature that we did not evaluate.

3) Zooming: Zooming in and out on specific folders is possible by clicking on or within a folder, allowing inspection of a sub-folder of particular interest.

4) User merging: It is possible to inform Git Truck which git usernames represent the same person. These will then be considered as a single user.

5) Hiding files and folders: Specific files, folders, or files of a certain type can be hidden from the visualization, in case they are irrelevant for the analysis (e.g. a committed package-lock.json file is larger than an entire project, a data folder that has 1K files, etc.).

6) Details on demand: Selecting the visual representation of a file or folder will display detailed information about it. This includes which authors has done the most work on it, when it was last changed, and more.

D. Usage

Git Truck is published as an npm package[14] to the npm registry. This approach gives a low barrier to entry, as it only requires the user to have Node.js installed, which is fairly common among developers.

There are several ways of installing and running Git Truck with the help of the npm package manager, the simplest being running the following on a system that has npx installed[15]:

```
npx git-truck@latest
```

This command downloads and installs the latest version of the package from the npm registry, then opens Git Truck in the default browser, and starts analyzing the given project. Once the analysis is done, the user is ready to investigate their system with the help of the UI presented in the previous sections.

[14]https://www.npmjs.com/package/git-truck

[15]npx is a tool bundled with Node.js, that can automatically download, install, and execute packages from the npm registry[16]

Given that with one command we allow the user to install and run Git Truck we believe that we have lowered the barrier of adoption to the minimum.

III. Architecture

One of the requirements that drives the architecture of the tool is the need to execute it locally, and thus, provide privacy for the user and assure them that their code is never uploaded anywhere. Indeed, privacy has been a reason for developers to be reluctant to upload their source code online[2].

A block diagram of the architecture of the system can be seen on figure 11. The various components in the diagram are discussed in the following subsections:

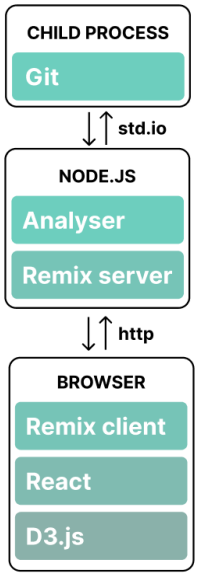

Figure 11. Git Truck Architecture.

A. Front-End and Back-End

When the user executes the git-truck npm package, a Node.js process is started, which serves up the user interface in the default browser on the host machine.

Since JS code running in the browser is not allowed to access the file-system, the repository analysis component is separated in it's own process with which the UI communicates. The UI communicates back to the analysis component of the tool via the full-stack framework Remix. Remix allows for easy state handling, such that when a user performs an action in the UI, for example clicking a button to hide a file, we can grey out the button, send a request to the analyser, and once the analyser sends a response, we can update the state in the UI, and reactivate the button.

B. The Analyzer

The analyzer is responsible for collecting structural and historical Git data from a repository and turning it into an intermediate data structure, that can be used by the visualization.

The analysis is done in two steps:

1) The file structure is gathered, via `git ls-tree`[17]
2) The commit history is processed. `git log`[18] is used to get the commit history.

To extract data from Git, the analyzer runs Git as a sub-process from Node.js. The decision of using git as a sub-process came after trying out git-specific libraries which had their limitations, e.g. having to be build locally on the users machine.

In our testing the sub-process spawning is very fast on Linux, but not on Windows or MacOS. However the performance hit is minimal, because the slowdown comes from starting and closing a process. On our Windows PC, spawning an empty process takes 6-13 ms, and we only spawn a small number of processes.

The resulting data is cached in a local file, and the full analysis will then only be done again, if changes are detected.

C. Chart layout generation

Git Truck uses algorithms from D3.js[19] to generate the layout of the bubble chart and the tree map. They use the included circle packing and treemapping algorithms, respectively.

To increase scalability, we modified the layout generated by D3.js, by showing folder names only when there is room for it, and not rendering elements that are too small to be seen. Rendering of the SVG graphics was implemented with React[20].

Transitional animations were implemented using the react-spring library[21], meaning that whenever the view changes, for example if the user zooms in on a folder, there will be an animation transitioning to the zoomed in folder.

IV. User Evaluation

We tested the usability of Git Truck to analyze our design choices and identify potential ways of improvements. In the following, we present the method and discuss the collected results.

A. Method

We adopted an asynchronous remote usability testing [7] approach. That is, participants were able to test Git Truck at different times and in their particular computer setups. We chose this method to allow participants to behave like they do in their daily work.

We conducted an initial pilot test by inviting personal contacts. Participants were invited to download, install, and use the features of our visualization tool to analyze a repository of their own. Additionally, we asked participants to fill in a questionnaire to collect their impressions. The results helped us to improve the questionnaire by clarifying questions that were, for instance, misunderstood or not interesting in terms of the research question.We improved the questionnaire. We

[17]https://git-scm.com/docs/git-ls-tree
[18]https://git-scm.com/docs/git-log
[19]https://d3js.org/
[20]https://reactjs.org/
[21]https://react-spring.io/

Table I
QUESTIONS THAT PARTICIPANTS HAD TO ANSWER WHEN TESTING THE USABILITY OF GIT-TRUCK.

Aspect	Question	Rationale
Setup	Did the tool open properly in your browser	To understand how smooth was the setup
User	Was anything about exploring the UI in the browser confusing? Was anything about setting up Git Truck confusing?	To identify confusing elements in the interface and configuration
Features	Which of the following features are you aware exist? Which of the following features did you find useful? If you found any of the features particularly useful, please explain how you used it	To examine how effective are the included features
Discoveries	Did you learn or discover something interesting about the repository? If yes, please describe it Did you discover anything about the repository, that made you want to change or rethink the code/folder? Did you gain any new insight into how people contribute to the repository?	To analyze how our tool promotes discoverability
Context	For how many years have you worked professionally with programming? What is your relation to the project? What kind of project is the repository releated to? How big is the team responsible for the repository How many commits are there in the project? How many files are there in the project?	To study the characteristics of the projects and development teams as well as the role of the participant
Views	Which chart gave you the best overview? If possible, please explain why? Which views did you find useful/interesting? If possible, please explain why it was useful to you? Which views did you not find useful? If possible, please explain your answer?	To understand how effective are the views and what features are particularly useful
Suggestions	Is there anything that you would like to see added/changed about the tool? How frequently do you think you would use Git Truck? Please explain If you have any further comments about Git Truck, please write them	To collect general impressions and suggestions from participants

aim to formulate questions that help us understand how users interact with Git Truck and how the tool helped them to learn about their software artifacts. Table I presents the list of questions that participants where asked when testing the usability of Git Truck.

Next, we invited a larger set of participants by sending direct invitations to computer science students and professional software engineers as well as advertising the evaluation to our contacts in social media (i.e., LinkedIn, Twitter).

B. Results

We collected results from 18 participants who answered the survey. The data gathered from the survey is available as supplemental material published on Zenodo[15]. In the following we organize the results of the evaluation by answering a series of questions.

Is the tool easy to install?

Yes. The setup process described in Git Truck's README[22] worked as intended for 86% of respondents [A]. Amongst the remaining ones, one reported a bug that we fixed; and another one used an uncommon distribution of Linux. In the end, all users that faced problems were able to install Git Truck by using `npm install` instead of `npx` [15, *S*]. The users largely thought that the UI was intuitive and easy to navigate, only with minor suggestions on how to clarify some elements [15, *Q*]. The setup could have been more minimal, if Node.js was not a requirement. That said, even for users not familiar with Node.js, the setup process went mostly without problems. [15, *R*].

Which views are perceived as most useful?

Although all views were considered fairly useful, *"Top contributor"* was the most liked (17/18) view by the users in the survey. Immediately after that is "Number of Commits" (13/18). People also found "File Extension" useful (12/18). "Last Changed" was considered the least useful (6/18) .

When it comes to layout, the bubble chart is the preferred diagram among the respondents, with 81% of users answering that the bubble chart provided a better overview [15, *I*]. The reasons for this are varied, and include "more intuitive to navigate when including subfolders", and "more organic, easier on the eye" ([15, *U*]).

The users that prefer the tree map, all have more than five years of professional programming experience, and they all noted that they prefer the efficient use of space of the tree map (e.g. "With many files, the bubble chart struggles to show the colors, while the tree map even when looking at the entire project shows more colors" ([15, *U*])).

What do users learn? *Code files written by only one author and the level of contribution by group members*. Indeed, many users were surprised by the distribution of authorship across files. One of them wrote:

> *"There were a few source code files that were ENTIRELY written by one author. I had known about some parts like "oh yeah, that was TOTALLY so-and-*

[22]https://github.com/git-truck/git-truck#readme

so's project", but I hadn't expected a few other files." [15, *M*]

Some students working on group projects were able to see if their group members did not contribute as much they should. A teacher reflected that Git Truck could be useful to encourage students to participate more in projects. He wrote: *"[...] to follow progress of a project. Maybe also to encourage students to participate more in a project"* [15, *P*].

Is there a kind of user or project for which the tool is more useful? *Novice developers who work on large collaborative projects*. The survey shows that only 37% of respondents (3/8) corresponding to developers who analyzed industrial repositories reported learning something new about their system with Git Truck. On the other hand, 80% (4/5) of the students who analyzed academic projects reported learning about the contribution of group members [15, *W*]

Individual responses show that students have the most opportunities to gain insight into a project's structure (e.g. *"found a mistake in the way one of our folders had been set up"*) and realize opportunities for improvement (e.g. *"Non-code files (pictures, models etc.) took up more of the repo than expected. It made me consider using something like LFS or hosting the files on a cdn"*). [15, *N*]

How often and in what context do the participants think that the tool could be useful? *Git Truck could be used at least once a month. Additionally, respondents believe it can be used in two main contexts: 1) by individuals to familiarize themselves with new repositories and 2) by teams to evaluate the progress on a project.* In particular, when asked how frequently the participants would use Git Truck, 38% answered that they would use it for occasional events, while 44% answered that they would use it on a monthly basis [15, *L*].

An educator noted that Git Truck could be used to follow progress on student projects. A professional programmer mentioned that Git Truck could be used to follow the progress of major refactorings [N]. Several users mentioned that Git Truck can be used to get familiar with a project:

> *"The tool seems useful to 'get a feel' about a project"*; *"The primary use cases [for the tool] I see are: Familiarising yourself with a new project, which is a one time thing. [...]"*

V. Discussion

a) Limitations of the sampling: The presented user study had a sample size of 18 respondents (22 including early responses), which was useful to get a general sense of, for who Git Truck is useful. To back these claims up empirically, a larger study could be made. This larger study would have to include research into how to get more people to respond to surveys. Our survey got out to a lot of people on social media, but only a fraction of them responded to the survey. Another way to gather data, would be to collect automatic usage data from Git Truck to, for example, see which views are being used most frequently.

b) Not everybody has npm installed: In the early phases of the study, we decided to build a local application, as described in section III. One person on Twitter, mentioned that they did not want to use the tool, because it involves installing Node.js, and that they would prefer to use an online tool. A future study could research how Git Truck as a website might coexist with Git Truck as a local app, and which kind of users would use which and why. A website might also be a better method of getting more people to respond to the survey, as the barrier to entry would be even lower. This study could also examine how willing users are to upload private repositories to online services. This was not researched, as we assumed that users with private repositories, did not want to share it with online tools, but this assumption might not be correct.

A user asked for Git Truck as a Docker image, such that they can run it without having to install npm [23].

c) Need for visualizing multiple projects at once: One respondent with professional programming experience, mentioned that Git Truck did not give them a good overview, because they wanted to see multiple repositories at once [15, *V*]. Git Truck might be more useful to companies using polyrepos if it showed multiple projects at the same time.

d) Comparison with GitHub Statistics: GitHub also has contributor statistics. They are "lower resolution" than GitTruck and they do not provide the same kind of hierarchical file-system focused overview, but it could be possible that some of the information presented by GitTruck could be gleaned also from there. Our intuition is that the "locality" of showing files in the context of their folders gives Git Truck an advantage over the simple statistics presented by GitHub, but one would have to do design a special experiment to evaluate this intuition.

e) Generalizability of the findings: We have designed Git Truck to be as easily usable as possible, and as simple as possible in order to be able to allow users to install it and try it on their own. Given it's simplicity, we believe that some of the results we discovered would be replicated by other similar hierarchical metric-enriched file-focused git visualization tools, in particular those pertaining to the expected frequency of use for such a tool, and the context in which such a tool could be beneficial. However, the limitations of the sampling discussed at the beginning of this section must be remembered.

f) Missing Telemetry: Some of the questions that we have asked, could have been answered by telemetry (e.g. most useful views, frequency of use for the tool, etc.). A data-based answer would be an important complement to the self-reporting that we describe in this paper. However, due to privacy issues we decided to not implement telemetry in the tool for this study. For the future we believe that telemetry would be a very valuable tool to use in a context of asynchronous remote usability testing like our's.

[23] https://github.com/git-truck/git-truck/issues/564

g) Bubbles vs. Treemaps: In the answers all the professionals preferred the tree maps. We believe that this is because professionals are looking at larger systems than students and this is why they prefer treemaps. In our own experiments, and also when generating the images for this paper, as soon as a system becomes very large, we prefer to switch to the treemap which makes better use of the limited space.

VI. Related work

Several works have inspired the design of Git-Truck. For instance, to create an insightful visualization, Git Truck displays multiple metrics at once and allows switching between different views. This approach is explored in greater depth by Michele Lanza and Stéphane Ducasse [17]. Wattenberger [28] demonstrates file structures visualized with circle packing, and several proposals for additional extensions relating to historical data. Among other things, she proposes a view to see which files were changed most recently, and which files have the most commits, both of which are implemented in Git Truck. A few other works focused on specific benefits of employing visualization (i.e., storytelling), features of versioning platforms (i.e., GitHub issues), and libraries (i.e., React). Kumar at al. [8] used visualizations for storytelling based on GitHub projects and Fiechter et al. [13] proposed *issue tale*, a visual narrative of the events and actors involved in GitHub issues. *React-bratus*, a visualization tool specific to React applications, is presented by Boersma and Lungu [4]. The tool enables the visualization of the component hierarchy of React applications. Similarly to Git Truck the expected audience of their tool are students and educators. *CorpusVis* [24] visualizes software metrics of Java software systems from the Qualitas Corpus. In contrast, Git Truck enables the visualization of software metrics from Git repositories.

Several studies have visualized Git repositories to analyze repository's evolution. To name a few examples, Elsen [10] proposed *VisGi*, a directed acyclic graph visualization for the analysis of branch structures, which is combined with a Sunburst visualization for branchs' contents. Andrea [1] introduced *UrbanIt*, which uses city visualizations to analyze software repositories evolution. North et al. [21] presented *GitVS*, a system that uses visualization and sonification for understanding the history of Git repositories. Feist et al. [12] presented *TypeV* that visualizes abstract syntax trees from Java source code to support the analysis of software evolution. Kim et al. [16] proposed *Githru*, an interactive visual analytics system that enables developers to understand the context of software development history through interactive visual exploration of Git metadata. *RepoVis* [11] and *Seesoft* [9] use a more information dense visual approach, by having each line in every file, represented and colored, for displaying metrics on a per-line basis. RepoVis sets itself apart by having a larger emphasis on showing the history and timeline of a file, and allowing full text search within file contents. RepoVis does not include an evaluation of the solution. In contrast, our investigation focus particularly on using hierarchical visualization techniques to provide users multiple views for the analysis of repository evolution.

A few other studies have visualized addressed collaboration. Schreiber [23] uses no-link visualizations of team members and external contributors to analyze collaboration in open-source systems. Malik [18] proposed a per file visualization that supports the analysis of authors' contributions to individual lines of code. The system uses two types of visualizations: a stacked bar chart for the analysis of dominant authors and a Seesoft-like plot for contributions aggregated by functions. Though these studies focus on the analysis of collaboration, as opposed to our work, they do not offer specific support for the analysis of repository evolution.

In summary, as opposed to previous works, Git Truck has been designed for the analysis of the Truck Factor of Git repositories. As a design study, Git Truck tackles a complex and relevant problem in software engineering by choosing well-known visualization techniques that have proven effective in other application areas.

VII. Future work

According to the study conducted, Git Truck is an overall easy-to-use tool, where users tend to appreciate the overview that it gives, about how people contribute to a project. The tool is mostly useful for academics, i.e. to students and teachers, allowing them to familiarize themselves with, and follow the progress of projects.

The primary use-case for Git Truck, according to the survey, is for students and educators to follow the progress of a project and better understand its structure. Several respondents mentioned that they envision Git Truck being useful for getting familiar with new projects [15, *M*], [15, *P*]. In the future we plan to explore this latter direction further with a user study specifically focused on problems during onboarding new developers on a project. We want to discover what views and sources of information can be integrated to better help with this process, while at the same time, keeping the tool language independent

Finally, adding telemetry and collecting actual usage data from users who opt-in for this, would allow us to even better understand the needs and the benefits for a tool like Git Truck.

References

[1] Ciani Andrea. "UrbanIt: Mobile 3D Git Visualization". PhD thesis. Università della Svizzera Italiana, 2015.

[2] Casper Weiss Bang and Mircea Lungu. "Codoc: Code-driven Architectural View Specification Framework in Python". In: *2021 Working Conference on Software Visualization (VISSOFT)*. IEEE. 2021, pp. 120–124.

[3] Christian Bird et al. "The Promises and Perils of Mining Git". In: *Proceedings of the Sixth Working Conference on Mining Software Repositories*. IEEE Computer Society, May 2009. URL: https://www.microsoft.com/en-us/research/publication/the-promises-and-perils-of-mining-git/.

[4] Stephan Boersma and Mircea Lungu. "React-bratus: Visualising React Component Hierarchies". English. In: *Proceedings of the 2021 Working Conference on Software Visualization (VISSOFT)*. United States: IEEE, 2021. DOI: 10.1109/VISSOFT52517.2021.00025.

[5] Caius Brindescu et al. "How Do Centralized and Distributed Version Control Systems Impact Software Changes?" In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 322–333. ISBN: 9781450327565. DOI: 10.1145/2568225.2568322. URL: https://doi.org/10.1145/2568225.2568322.

[6] Frederick Brooks Jr. "No Silver Bullet Essence and Accidents of Software Engineering". In: *IEEE Computer* 20 (Apr. 1987), pp. 10–19. DOI: 10.1109/MC.1987.1663532.

[7] Anders Bruun et al. "Let your users do the testing: a comparison of three remote asynchronous usability testing methods". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 2009, pp. 1619–1628.

[8] Shishir Dubey et al. "Data visualization on GitHub repository parameters using elastic search and Kibana". In: *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE. 2018, pp. 554–558.

[9] Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner. "Seesoft-A Tool For Visualizing Line Oriented Software Statistics". In: *IEEE Trans. Software Eng.* 18 (1992), pp. 957–968.

[10] Stefan Elsen. "Visgi: Visualizing Git branches". In: *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE. 2013, pp. 1–4.

[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] Michael D Feist et al. "Visualizing project evolution through abstract syntax tree analysis". In: *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE. 2016, pp. 11–20.

[13] Aron Fiechter et al. "Visualizing Github issues". In: *2021 Working Conference on Software Visualization (VISSOFT)*. IEEE. 2021, pp. 155–159.

[14] A. N. Habermann, Lawrence Flon, and Lee Cooprider. "Modularization and Hierarchy in a Family of Operating Systems". In: *Commun. ACM* 19.5 (May 1976), pp. 266–272. ISSN: 0001-0782. DOI: 10.1145/360051.360076. URL: https://doi.org/10.1145/360051.360076.

[15] Thomas Kilbak et al. *Git Truck supplemental material*. Zenodo, June 2022. DOI: 10.5281/zenodo.6769782. URL: https://doi.org/10.5281/zenodo.6769782.

[16] Youngtaek Kim et al. "Githru: visual analytics for understanding software development history through git metadata analysis". In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2020), pp. 656–666.

[17] Michele Lanza and Stéphane Ducasse. "Polymetric Views-A Lightweight Visual Approach to Reverse Engineering". In: *Software Engineering, IEEE Transactions on* 29 (Oct. 2003), pp. 782–795. DOI: 10.1109/TSE.2003.1232284.

[18] Saad Malik. "Git repository visualization: visualizing file authorship and dominance". In: (2017).

[19] Leonel Merino et al. "A Systematic Literature Review of Software Visualization Evaluation". In: *Journal of Systems and Software* 144 (June 2018). DOI: 10.1016/j.jss.2018.06.027.

[20] Christopher R. Myers. "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs". In: *Physical Review E* 68.4 (Oct. 2003). DOI: 10.1103/physreve.68.046116. URL: https://doi.org/10.1103%2Fphysreve.68.046116.

[21] Kevin J North, Anita Sarma, and Myra B Cohen. "Understanding Git history: A multi-sense view". In: *Proceedings of the 8th International Workshop on Social Software Engineering*. 2016, pp. 1–7.

[22] Rolf-Helge Pfeiffer and Andrzej Wąsowski. "The Design Space of Multi-Language Development Environments". In: *Softw. Syst. Model.* 14.1 (Feb. 2015), pp. 383–411. ISSN: 1619-1366. DOI: 10.1007/s10270-013-0376-y. URL: https://doi.org/10.1007/s10270-013-0376-y.

[23] Andreas Schreiber. "Visualization of contributions to open-source projects". In: *Proceedings of the 13th International Symposium on Visual Information Communication and Interaction*. 2020, pp. 1–2.

[24] Jack Slater et al. "CorpusVis–visualizing software metrics at scale". In: *2019 Working Conference on Software Visualization (VISSOFT)*. IEEE. 2019, pp. 99–109.

[25] *Stack Overflow Developer Survey 2021*. URL: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-other-tools. (accessed: 2022-05-13).

[26] M-A Storey, Casey Best, and Jeff Michand. "Shrimp views: An interactive environment for exploring java programs". In: *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*. IEEE. 2001, pp. 111–112.

[27] Sergi Valverde and Ricard Sole. "Hierarchical Small-Worlds in Software Architecture". In: *Dynamics of Continuous Discrete and Impulsive Systems: Series B; Applications and Algorithms* 14 (Jan. 2007), p. 1.

[28] Amelia Wattenberger. *Visualizing a Codebase*. 2021. URL: https://githubnext.com/projects/repo-visualization. (accessed: 2022-05-09).