Use the Forks, Look! Visualizations for Exploring Fork Ecosystems

Siyue Chen*, Loek Cleophas*†, Sandro Schulze‡, and Jacob Krüger*
*Eindhoven University of Technology, The Netherlands
{ s.chen | l.g.w.a.cleophas | j.kruger } @tue.nl
†Stellenbosch University, Republic of South Africa
‡Anhalt University of Applied Sciences, Germany
sandro.schulze@hs-anhalt.de

***Abstract*—Forking is a common practice in open-source and industrial software development, leading to the emergence of complex fork ecosystems. Understanding the evolution and relationships within such ecosystems is crucial for developers and project managers to ensure that useful changes are merged back into the original project or synchronized between forks. However, understanding complex fork ecosystems with up to tens of thousands of forks in different states (e.g., abandoned, co-evolving) is challenging, with visualizations being a means to address this challenge. In this paper, we investigate six visualizations designed to provide key insights into the dynamics of fork ecosystems. We started our work by analyzing GitHub community feedback on the official Network Graph visualization for fork ecosystems and categorized the fork-related tasks mentioned by developers in 237 comments. Then, we designed our visualization prototype VisFork, which contains six different visualizations that serve the three most frequently mentioned tasks. These visualizations allow users to explore temporal patterns, commit classifications, and collaboration dynamics across a fork ecosystem. Through a user study involving 10 GitHub community participants and seven students, we evaluated the usefulness of the visualizations. The results demonstrate the potential of VisFork to provide valuable insights into fork ecosystems, with positive feedback on the visualizations, but also suggestions for further improvements. Our work contributes to the development of user-centered tools that help to understand the intricacies of fork-based development and promote collaborative software-development practices.**

***Index Terms*—Fork Ecosystems, Software Repositories, Visualizations, Variant-Rich Systems, Software Evolution**

I. Introduction

Forking is the process of creating a copy of a repository and using that copy to implement independent changes without affecting the original repository; and without the need for permission from the owner of that repository [8], [10]. Fork-based development has become a primary means for extending and evolving a system or for developing a variant of that system in open-source communities as well as industry [15], [16], [18], [21], [35], [42]. Due to the independence of forks, developers can work in parallel on a project and maintain their own system variants. So, forking enables collaboration within a team as well as contributions from outside the team [12], [37]. Specifically, a forked repository has its own history and any changes made in the fork do not affect the original repository. Still, the fork developers can submit their changes back to the original repository through a *pull request*, which allows the maintainers of the original repository to review the changes and merge them into their project [11], [12]. Note that branching allows to create copies of a system *within* a repository, while forking creates a logically independent project that may never be merged back into its parent.

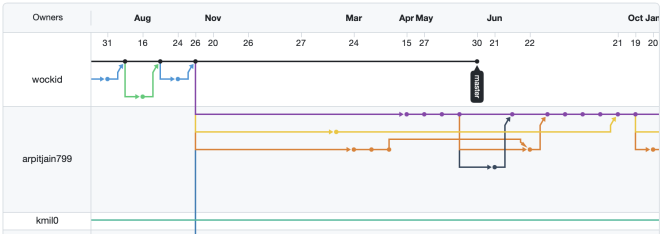

Fig. 1. An example for a Network Graph on GitHub. https://github.com/wockid/Adafruit_Python_DHT/network

Despite the benefits, it is challenging to track the decentralized development in forks, and the difficulty grows as the number of forks increases [16], [18], [21], [29], [35], [39]. A larger number of forks (e.g., 49,400 for the Linux Kernel, 18,500 for the Marlin 3D printer firmware) results in a *fork ecosystem*, a special type of a *variant-rich system* [17], [34], and can cause severe problems. For instance, development and testing efforts may be redundant, contributions get overlooked, or it becomes impossible to fully understand the whole ecosystem [5], [12], [29], [41], [42]. Visualizations are one means to obtain a representation of the relationships between forks, making it easier to analyze fork ecosystems and mitigate the aforementioned problems.

The simplest way to visualize the history of one repository is to display its evolution graph. For instance, developers can use the command-line tool `git log`, which shows a list of all commits made to the repository along with information about the author, date, and commit message. Unfortunately, it displays information in plain text—making it difficult to use for repositories with many commits—and does not consider forks. Thus, more advanced visualizations have been proposed to provide a more comprehensive and user-friendly view of a repository's history and its potential fork ecosystem.

The most prominent of such visualizations may be GitHub's *Network Graph*,[1] which shows the fork ecosystem of a repository ordered by commits (bullets). As we display in

[1]https://docs.github.com/en/repositories/viewing-activity-and-data-for-your-repository/understanding-connections-between-repositories

Fig. 1, a Network Graph visualizes how a fork ecosystem evolves along a timeline (top axis), when a fork is created (outgoing arrows with new color), and when forks are merged back into another (incoming arrows)—thereby indicating how active a fork is. While the Network Graph has been acknowledged for its basic visualizations, the GitHub community has extensively discussed its limitations,[2] emphasizing the need for visualizations that allow for more in-depth insights into fork ecosystems. Among the 237 replies (until February 14, 2023), 90 % of the discussants indicate that they use the Network Graph at least once a day. The major usages (we provide a more detailed analysis in Sec. IV-A) include

1) exploring fork histories and relations,
2) discovering the activity status of forks, and
3) identifying work in forks.

Using the Network Graph, developers can start discussions with collaborators and make decisions about their projects. However, they also note various limitations of the current visualization:

1) it shows only forks that are directly connected to the main repository and not forks of forks;
2) it can be difficult to understand the relationships between different forks and branches, especially if they have complex naming conventions;
3) it does not allow users to filter or search for specific forks, and thus is difficult to navigate; and
4) it does not show the differences between forks that are actively maintained and those that are stale or abandoned.

Other concerns noted are related less to the actual visualization, but more to performance improvements or minor features.

The active discussion and extensive feedback by the GitHub community clearly indicate the practical need for improving visualizations for fork ecosystems. In this paper, we advance in this direction by implementing and comparing different visualizations for tackling the above limitations. Specifically, we have implemented a prototype, VisFork, with six different visualizations that provide an overview of a fork ecosystem, display deviations, visualize its evolution, and deliver insights into the development within forks. We evaluated the feasibility of our visualizations through a user study with 10 practitioners who participated in the GitHub discussion and with seven students. Our results indicate that the visualizations are helpful for software developers, even though they also indicate potential improvements for future work.

In more detail, we contribute the following in this paper:

- We present six visualizations for fork ecosystems that can help developers facilitate their work.
- We describe and discuss the results of our user study, providing insights into the feasibility of the visualizations.
- We publish our prototypical implementation VisFork in an open-access repository for others to reuse.[3]

Overall, we contribute insights into how different visualizations can help developers analyze fork ecosystems. We also highlight potential research directions that can build on our contributions.

[2]https://github.com/orgs/community/discussions/40469

[3]https://zenodo.org/doi/10.5281/zenodo.10462693

II. Related Work

Researchers have extensively studied fork ecosystems and their evolution, covering various issues, such as redundant development, fork integration, or feature identification [5], [8], [10]–[12], [14], [16]–[18], [20], [21], [29], [31], [33]–[35], [37], [39], [40], [42]. These works highlight the complexity of fork ecosystems and motivate the need for corresponding visualizations. Moreover, three existing literature reviews provide overviews on visualizations for variant-rich systems [3], [25], [27], with fork ecosystems representing a special type of such systems. Based on these reviews and our knowledge, we next summarize the closest works connected to our own.

A common way to visualize the changes in a fork is to structure commits in the order in which they occur (e.g., as in the GitHub Network Graph). Unfortunately, commits may have unreliable descriptions and can contain large or independent changes [1]. So, a graph of commits may not be very expressive about the types and numbers of changes that occurred within a fork. Aiming to tackle such problems, Ren et al. [30] have proposed a web-based solution to visualize forks in a table. They use natural language processing to extract keywords from source code, comments, and commit messages. Then, a user can search related forks of a GitHub repository, tag the forks with labels, and get a simple overview based on the assigned keywords. Such a visualization is useful for developers to investigate whether a certain feature has been implemented to avoid duplicated work or to explore trends over time across forks. Similarly, Zhou et al. [40] summarize the work that has been done within forks in tabular form, specifically aiming to identify what features have been implemented. Unlike other concepts like charts or graphs, regular tables do not provide an intuitive visualization for identifying patterns and relationships between forks. While we have been inspired by these works, we integrate multiple visualizations and evaluate their value via a user study, which has not been the focus of such prior works.

Biazzini and Baudry [2] present tooling for analyzing commit histories of fork ecosystems on GitHub. To build their visualizations, they create an “umbrella repository” that consists of an original repository and all its forks. Then, they compute dispersion metrics based on commits to quantify the degree of dispersion between the forks and the original repository. Biazzini and Baudry visualize the forks as a Chord diagram generated via Circos [22], which can visualize multiple pieces of information at once within a circular layout, for instance: The order of elements (forks) indicates the creation time of forks, the width of these elements the number of non-unique commits in the fork, and colors represent forks with the same number of commits. Using a Chord diagram, it is easy to distinguish how a project is dispersed among forks at a high level, which Biazzini and Baudry used to derive collaboration models. However, this visualization is loaded with information that may overwhelm a user, it focuses on metrics but not in-depth insights, and it has not been evaluated empirically. We complement this previous work by investigating different visualizations and their feasibility for actual developers.

Duszynski et al. [6], [7] have proposed set-based comparisons of software variants and forks. In particular, they define a technique to construct a set model that contains sets at every level of the system's structure, from files to folders. Then, they apply a comparison function to determine the content similarity of the sets, which are then visualized. For instance, a structured tree diagram can be used to show how much content forks share based on color intensity. Alternatively, a tree-map set diagram can visualize how many lines of code are core, shared, or unique in each fork. While the visualizations of Biazzini and Baudry as well as Duszynski et al. are improvements on widely established visualizations like Venn diagrams, their readability still suffers when comparing a larger number of forks. As a result, the visualizations may be unsuitable for developers to get an overview of all forks and they do not shed light on the actual evolution or work performed. Our contributions are complementary and provide novel insights into visualizations, providing empirical insights into which ones can help developers to analyze fork ecosystems.

III. Goal and Research Objectives

The goal of our work is to propose visualizations that facilitate the analysis and understanding of fork ecosystems. Our motivation stems from our own work on fork ecosystems [16]–[18], [20], [21], [29], [33]–[35], related work on fork ecosystems (cf. Sec. II), and practitioners' experiences (cf. Sec. IV-A). Based on such sources, we consider visualizations for fork ecosystems key for exploring these, be it for research or practice.

To scope our work in this paper, we have defined three Research Objectives (ROs) based on our motivation as follows:

RO_1 Identify what properties of a fork ecosystem are important for developers to visualize.
To design helpful visualizations, it is vital to learn what properties of a fork ecosystem are important for developers to learn about, and thus likely for researchers to investigate. So, our first objective was to identify the properties of fork ecosystems we would present with our visualizations. Building on our findings (cf. Sec. IV-A), we started to design our visualizations—and other researchers can use these findings in the future.

RO_2 Design visualizations that help developers and researchers understand the identified properties.
Our primary goal for this work was to design helpful visualizations, that is visualizations that facilitate the comprehension of fork ecosystems. For this purpose, we designed a set of visualizations that are feasible for the properties identified and we implemented a prototype (cf. Sec. IV-B). For the benefit of practitioners and researchers, we share our prototype in an open-access repository.[3]

RO_3 Evaluate to what extent the visualizations help understand fork ecosystems.
Lastly, we investigated to what extent our visualizations help understand fork ecosystems. To tackle this objective, we performed a user study with 10 experienced GitHub community members and seven students (cf. Sec. V), asking them to answer different questions, rate our visualizations, and compare them against the GitHub Network Graph. The participants' responses underpin the value of our visualizations, and sketch how to improve them in the future to facilitate fork-based development.

Addressing these objectives provides helpful visualizations for future research and for facilitating developers' work on forks.

IV. Visualizing Fork Ecosystems

Next, we report our analysis of a discussion among GitHub community members regarding GitHub's Network Graph. Based on this analysis, we identified design goals for our visualizations, aiming to tackle the prevalent limitations of the Network Graph that the developers perceive. Then, we proceed with the details of our visualizations. Specifically, we discuss each visualization we adopted, reason about their selection, and explain how they help address the limitations we identified. In the end, we describe our prototypical implementation, VisFork, and report on the challenges we encountered during our work, providing insights for other researchers and developers.

A. Defining Design Goals (RO_1)

As we exemplify in Fig. 1 and Sec. I, GitHub offers a visualization for fork ecosystems, the Network Graph. This graph displays a timeline of commits including the 100 most recently pushed forks. GitHub has also collected feedback from its community members regarding the use of the Network Graph, aiming to understand how these use the graph and what could be improved. This feedback has been collected in the form of unstructured text in an open discussion[2] in which 237 comments have been posted by GitHub users.

We took this ideal opportunity to build on an elaborate discussion of a part of our target population (developers working on fork ecosystems) on their demands for novel visualizations (our research goal). To analyze the comments, the first author of this paper manually inspected each comment following an open-coding process. Specifically, the first author read each comment and coded *what properties of a fork ecosystem the developers explore via the Network Graph* as well as *what additional properties they would like to explore*. For instance, a community member mentioned that they hoped to "*Check for any fork that is more up-to-date than the original repo.*" We coded this comment to be about *discovering the best/latest fork*, since it showed the intention to identify forks that are more up-to-date. Note that a comment could involve various tasks, which is why the numbers in Tbl. I do not add up to 237 (100 %). Through this process, we obtained a list of properties that developers investigate and would like to investigate using visualizations—both of which a new visualization should aim to address. After the coding, the first author performed an open-card sorting to classify the individual codes into common categories. This process and the consequent data were continuously checked by two other authors during multiple discussions, improving our understanding of what features the developers ask for and how existing visualizations should be extended.

In Tbl. I, we provide an overview of our classification. Most of the comments relate to *exploring histories and*

TABLE I
CLASSIFICATION OF THE 237 GITHUB COMMENTS.

category	task	mentioned	
		#	%
explore history and relations	explore branch activity	90	38
	explore fork activity	71	30
	explore task assignment	52	22
	explore commit activity	38	16
	explore project structure/history	28	12
discover forks' status	discover active forks	64	27
	discover best/latest fork	26	11
	discover inactive forks	20	8
	discover outdated forks	12	5
identify work in forks	identify hidden work	50	21
	identify improvements/bug fixes	28	12
	identify features	20	8
	identify diverging directions	2	1
inspect dependencies	inspect fork dependencies	3	1
	inspect branch dependencies	2	1

relations, while *inspecting dependencies* has been mentioned the least. Particularly, the developers use the Network Graph to *explore branch activity*, indicating a frequent demand for investigating the histories and relations of branches in their projects. Interestingly, this category as well as *discovering forks' status* and *identifying work in forks* are not only common use cases and demands by developers, but also studied or identified as problems in research [5], [14], [29], [31], [35], [40]. This underpins that the research in this direction tackles a relevant and important problem, and visualizations are a means to help not only practitioners but researchers as well. Particularly, besides *explore branch and fork activity*, the developers are interested in *discover active forks*, *explore task assignments*, and *identify hidden work* in other forks. While there seems to be a common need for improving visualizations for the first three categories in Tbl. I, *inspect dependencies* has rarely been mentioned, indicating that this is not a primary property developers want to explore via the Network Graph.

Based on such insights, we defined three Use Cases (UCs) that guided the design of our visualizations and our evaluation:

- UC_1 *A project manager, developer, or researcher wants to explore the ratio of active forks in an ecosystem.* According to our data, the developers are most often interested in *exploring the histories and relations* in a fork ecosystem, particularly in *exploring fork activity* as the second most commonly mentioned task (71, 30 %). This indicates a strong interest in gaining a general overview understanding of fork activity in an ecosystem.
- UC_2 *A project manager, developer, or researcher wants to discover an active fork of an abandoned repository.* Besides exploring activity within an ecosystem, the developers also need to *discover the status of a specific fork*, with *discovering an active fork* (64, 27 %) being the most common task in this category and third overall. This indicates a need for visualizations that can speed up the identification and analysis of active forks, particularly if these were forked from now abandoned repositories (i.e., outdated or inactive forks).
- UC_3 *A project manager, developer, or researcher wants to identify a hidden feature that has not been pushed to the original repository.* When exploring fork ecosystems, developers are often interested in *identifying hidden work* within forks (50, 21 %). This demand substantiates the need for a visualization that aids in discovering and integrating hidden features from forks.

Note that we focused on the most commonly mentioned, but also different, categories and tasks that relate to forks (and not branches) to specify these use cases for our visualizations.

B. Designing the Visualizations (***RO₂***)

To design visualizations for our use cases, we first derived what information would be needed for a user to actually fulfill each use case (e.g., a fork's status for UC_2). Then, we analyzed what visualizations could be feasible to display such information. For this purpose, we studied literature reviews on the matter [3], [25], [27] as well as similar related work (cf. Sec. II), and summarized what visualizations have been used for similar information. Afterwards, we adapted, combined, and extended these visualizations to make them fit the use cases and the developers' demands we identified in Sec. IV-A. Finally, we implemented, tested, and refined our prototype VisFork. We want to note that we chose the six visualizations we ended up with because we considered them to provide complementary, valuable, and helpful insights into our use cases; yet a subset of these, or additional or different visualizations may be even better suited for the same purposes. So, our user study (cf. Sec. V) has the goal of contributing evidence on which of the visualizations are helpful for developers.

In Fig. 2, we display an overview of VisFork, which is an integrated and interactive dashboard comprising six visualizations. We implemented our prototype using D3.js,[4] a framework for creating interactive and dynamic data visualizations. Also, we integrated VisFork with the GitHub API to retrieve data about forks. Next, we introduce the individual visualizations, explaining their design and why we consider them relevant.

Date Range Slider. As the first element (① in Fig. 2), VisFork presents a bar chart that displays the distribution of commits over the queried time range. This interactive chart provides a visual representation of the number of commits (i.e., changes) in a fork ecosystem over time, serving as a date range slider that users can adjust to specify what time period they want to analyze. Bar charts are an excellent visualization for this purpose, since they can clearly represent quantitative data across different categories—in this case, the number of commits over time. The x-axis of the bar chart represents the timeline, spanning from the earliest to the latest commit in the queried fork ecosystem. On the y-axis, VisFork represents the number of commits, with the height of each bar corresponding to the number of commits made during a specific time period. So, the bar chart provides an immediate sense of the ecosystem's activity over time, highlighting periods of intense development and periods of relative inactivity (UC_1).

[4]https://d3js.org

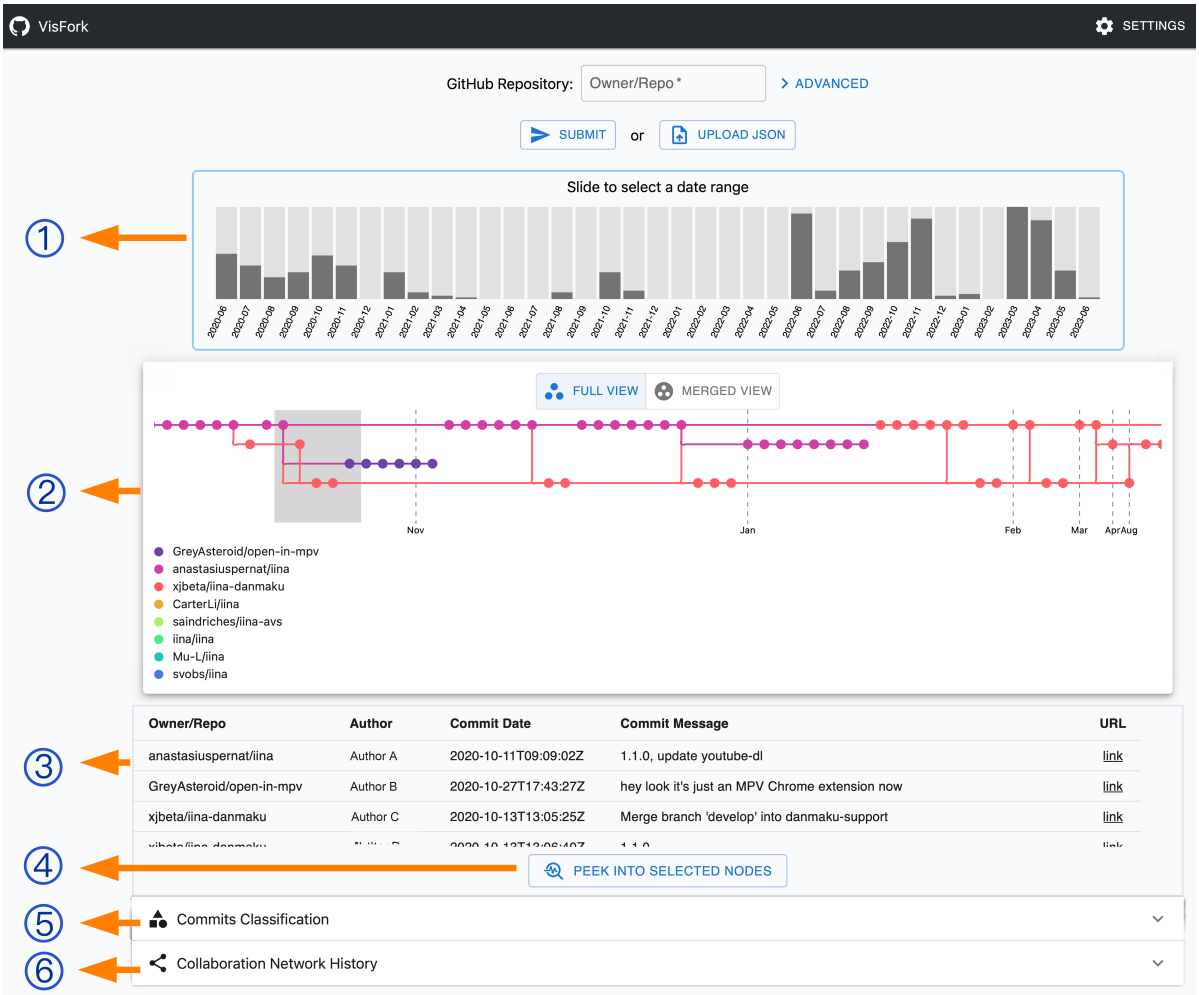

Fig. 2. Overview screenshot of VisFork for a smaller, real-world fork ecosystem (author names are anonymized): ① date range slider; ② full commit timeline; ③ commit detail list; ④ word cloud; ⑤ commit classification tab; and ⑥ collaboration network history tab.

The bar chart has a brushing feature, allowing a user to select a time range by clicking and dragging over the desired bars. Once a time range is selected, the chart and all subsequent visualizations update to reflect only the data within this range. This feature enables users to focus their analysis on periods of particular interest, providing a more tailored view of the repository's development history (UC_1, UC_2).

Commit Timeline. Next, VisFork provides a detailed view on the fork's commit histories (② in Fig. 2). More precisely, we offer two views: a typical *full view* and a *merged view*. Both views are inspired by well-established representations for software evolution (e.g., GitHub's network graph), and we used a similar Sugiyama-style graph drawing for the layout. The Sugiyama style, also known as layered graph drawing, is particularly effective in representing nodes (commits) and edges (connections between commits, and thus forks), while minimizing crossings and clearly delineating levels or ranks (time progression). Each lane in the graph corresponds to a fork over a time period without diverging behavior, and each node represents a commit to that fork. This layout provides a clear, chronological view of the commit history, making it easy for users to trace the forks' evolution over time (UC_1, UC_2).

In Fig. 2, we show the *full view*, which displays all fetched commits. This default visualization provides the well-known and comprehensive overview of the commit history of the forks similar to the Network Graph and other tools. This view allows users to explore every commit made to a fork, providing a complete picture of the development history (UC_1, UC_3). To distinguish between forks, each fork is assigned a unique color. Such a color-coding allows users to easily trace the commit history of each fork on the timeline.

We added the *merged view*, which simplifies the timeline by visually merging commits that have not been merged or forked from another fork. Consequently, this view shows only the nodes representing points of divergence. This provides a clearer picture of the evolution in a fork ecosystem, highlighting when forks are created and merged, removing any other information (UC_2, UC_3). In Fig. 3, we display an example for this merged view, which corresponds to the example in Fig. 2. We use two notations to distinguish nodes in the merged view: a circular node represents a single commit, while a square node represents multiple combined commits.

Lastly, the commit timeline has two more features. First, it allows to quickly inspect a commit by hovering over the

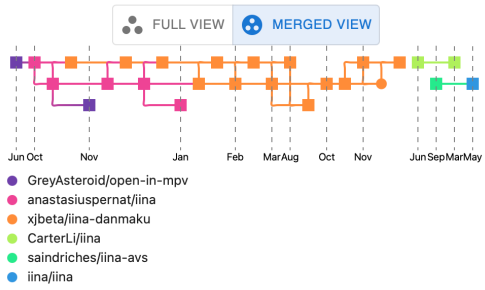

Fig. 3. Example for the merged view.

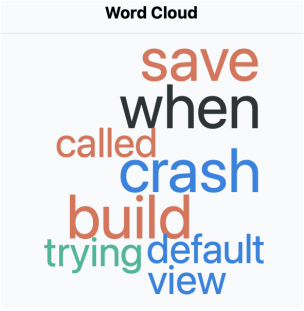

Fig. 4. Example for a computed word cloud.

respective node. An overlay box displays the metadata of the commit, including the owner, the repository name, the branch name, the commit ID, and the submission time. This feature allows users to gain a peek at each commit without leaving the visualization (UC_3). In addition, when a user clicks on a node, they are redirected to the corresponding commit page on GitHub, so that they can access all details of the commit. Second, a brushing feature allows a user to select specific nodes for further analysis. By clicking and dragging over the desired nodes, a user can select a subset of the commits. Once these nodes are selected, detailed visualizations on these commits are displayed in the following visualizations. This interactive feature enables users to focus on commits of particular interest.

Commit Detail List. After the commit timeline, VisFork presents a commit detail list in the form of a table (③ in Fig. 2). This table is inspired by related work [30], [40] and provides a structured overview of the detailed, multi-dimensional commit data of the nodes selected in the commit timeline. Tables are a well-established and comprehensible visualization for displaying such information in a structured format. Each row in the table represents a single commit, while each column represents a different attribute of the commit.

The table includes five columns, each providing a specific piece of information about a commit. First, the column *Owner/Repo* lists the owner and the name of the repository. Second, the column *Author* shows the author of the commit. Third, the column *Commit Date* displays the date and time when the commit was made. Fourth, the column *Commit Message* provides the message associated with a commit, which ideally includes a brief description of the changes made in the commit. Finally, the column *URL* includes a hyperlink to the commit page on GitHub. The tabular layout allows users to easily analyze these attributes for different commits, aiding in the identification of patterns and anomalies (UC_3). We provide a button below this table to allow a user to dig into the details of the selected nodes based on the following visualizations.

Word Cloud. Using a word cloud (④ in Fig. 2), we aim to provide a high-level overview of the work done within the selected nodes (UC_3). After a user clicks on *Peek into selected nodes*, VisFork shows a popup that displays a word cloud generated from the commit messages of the selected nodes. We show an example in Fig. 4. To generate the word cloud, we first extract all words from the commit messages of the selected nodes. Then, we count the frequency of each word, discarding common stop words, such as “the” or “and.”

Word clouds are particularly effective for text analysis, because they visually represent the frequency of words in a given text. In this case, we summarize the selected commit messages to shed light into what the respective developers have implemented. The more frequently a word occurs in the commit messages, the larger it appears in the word cloud, allowing a user to quickly grasp the most common themes associated with the selected commits. Additionally to scaling sizes, the word cloud also uses colors to represent the part of speech of each word. For example, verbs are represented in orange, while nouns are represented in blue. By associating colors with specific parts of speech, a user can quickly gain insights into the focus of the respective development activities. For instance, the colors can help identify whether there are more discussions about bug fixes (verbs) or documentation improvements (nouns). In summary, the Word Cloud provides a quick and intuitive overview of the work done in the selected nodes, making it an effective tool to analyze commit messages. By using size to represent word frequencies and color to represent parts of speech, the word cloud offers a multi-dimensional view on the commit messages.

Commit Classification. Swanson [36] has proposed three categories to classify software-maintenance activities that are often reused in research [13], [19], [24], [28], [38]:

Adaptive Changes modify the software to new environments or requirements, for instance, via dependency updates, compatibility improvements, feature additions, or migrations to a new operating system.

Corrective changes fix errors or bugs in the software, for instance, by addressing a bug report.

Perfective Commits improve the performance or maintainability of a system, for example, via performance improvements, documentation clarifications, or refactoring.

Classifying commits into these categories can provide valuable insights into the development processes exhibited within a fork ecosystem (UC_2, UC_3). For example, a high proportion of corrective commits may indicate issues with software quality, while a high proportion of adaptive commits may suggest a rapidly changing environment or requirements. In other words, understanding the distribution of commit types can help users identify areas a fork focused on as well as potential issues in the development processes.

Deep learning models have been explored to classify commits into the three categories [9], [24], [26], [32], but the nature of natural-language commit messages challenges their

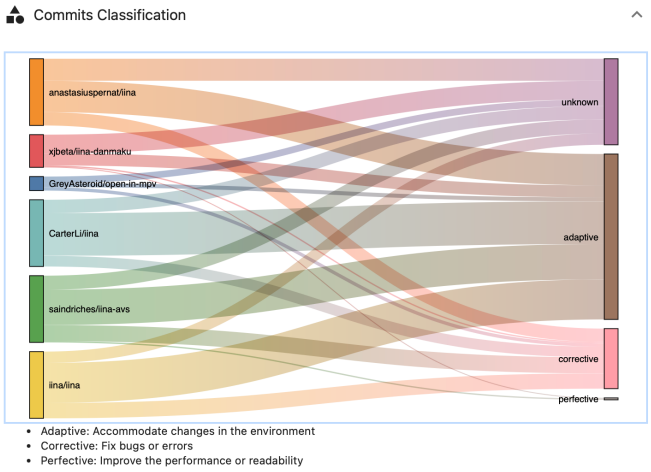

Fig. 5. Example for the commit classification tab.

performance. We therefore adopt a simple model instead: we match commit messages against stemmed core terms provided by Choshen and Amit [4]—which can also be executed faster. Some example core terms are:

Adaptive: configure, extend, support, enable
Corrective: bug, fix, fail, defect, error
Perfective: document, readme, refactor, typo, style

If a commit message matches core terms from multiple categories, we assign the category with the most matches (and the first one in case of a draw). Conversely, if we find no match, we categorize a commit as unknown. As a concrete example, a commit message reading "*Fixed shortcuts and update documentation for HistoryWindowController.*" matches two core terms: "fix" and "documentation". Since both matches occur once and "fix" occurs first, the commit would be categorized as a corrective commit.

Using this model, we avoid performance overheads and can execute the commit classification on-demand. Also, this classification is hidden in a separate tab by default, but is easily activated by clicking on the tab. Upon clicking, the tab expands to display a Sankey diagram like the one we display in Fig. 5. A Sankey diagram is particularly useful for visualizing the distribution of a quantity (commits) across different categories (adaptive, corrective, and perfective). On the left side of the Sankey diagram, we display the forks and on the right side we classify their commits. The width of each flow in the diagram is proportional to the quantity it represents (i.e., the number of commits), allowing users to quickly grasp the relative proportion of each type of commit and understand what a fork was created for (e.g., feature fork, bug-fixing fork).

Collaboration Network History. The last visualization we have designed is a graph to depict the evolving relationships between contributors and forks over time (⑥ in Fig. 2). In fact, this graph is a dynamic representation that highlights how contributions and collaborations change as the fork ecosystem evolves. Graphs are well-suited for depicting these relationships, since they use nodes to represent entities and edges to represent relationships. In our collaboration network history, each node represents a contributor or fork, and each edge represents a

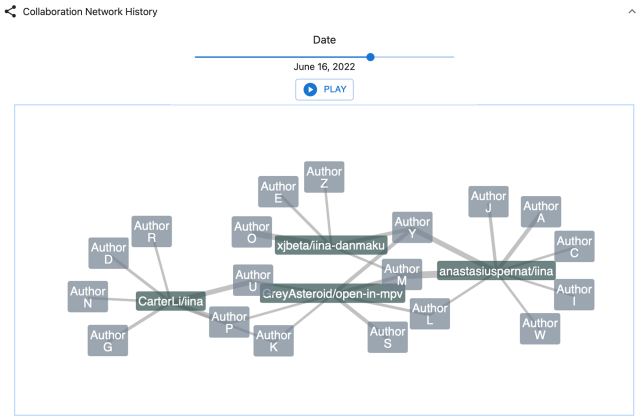

Fig. 6. Example for a collaboration network history (anonymized).

contribution to a fork. Moreover, graphs are flexible and can easily accommodate changes in the data. This is particularly important in our case, in which the relationships between contributors and forks change over time. As new contributions are made or new forks are created, new nodes and edges must be added to the graph, evolving the graph along the fork ecosystem. This allows users to easily see who contributed to which forks and how these relationships evolve over time (UC_2).

Just as the commit classification, we have designed the collaboration network history to be available on-demand. Initially, this visualization is hidden within a tab, too. After clicking on the tab, it unfolds and displays the visualization, for which we show an example in Fig. 6. We can see that the graph is centered around forks (in green), with nodes representing contributors (in gray) and edges their contributions to the different forks. A key feature is the date slider that allows a user to view the state of the collaboration network at any day of its history. Also, if a user presses the button *Play*, the date will increase automatically to show how the network evolves. This visualization provides insights into how the fork ecosystem and its community evolved, allowing to understand temporal dynamics of collaboration in the fork ecosystem.

V. User Study ($\mathbf{RO_3}$)

We evaluated whether our visualizations were reasonable by conducting a user study with two groups of participants: GitHub Community members (Sec. V-A) and students (Sec. V-B). To design the user studies, questions, and tasks, the first author proposed an initial version that all other authors tested and reviewed via pilot runs. After multiple iterations and refinements, we finalized a version of the user study that captured the most relevant parts while limiting the time required by participants to complete it. The whole user study was reviewed by a Data Steward specialized in software engineering and received approval from the Ethics Review Board of Eindhoven University of Technology. All participants had to sign a consent form and could stop at any point without consequences, and we share their anonymous responses in a persistent open-access repository.[5] In the following, we describe the details of our user study.

[5]https://doi.org/10.5281/zenodo.8199719

A. GitHub Community Participants

To gather insights on the practical feasibility of our visualizations, we identified a pool of potential participants from the 237 GitHub users whose comments we analyzed (cf. Sec. IV-A). Specifically, we manually reviewed their public GitHub profiles and identified 92 users who shared their email addresses publicly. Note that we invited only GitHub users that engaged on the topic (i.e., participated in the discussion) and that shared their e-mail publicly for contacting, to avoid spamming uninterested developers. We sent one mail (no reminders) to each of the 92 users, motivating our study and inviting them to test our prototype. Involving practitioners engaged with the topic in our study improves our confidence that their feedback is based on extensive experiences and honesty. Due to their experience, we did not define concrete tasks, but asked the GitHub users to explore our prototype (e.g., using provided data containing a small sample of eight forks with 5,684 commits spanning three years or an own fork ecosystem) and to subsequently complete an anonymous survey.

Our survey included background questions about a participant's experiences with version control systems and fork-based development, how many forks they would be interested to visualize, as well as what properties they would like to analyze. Then, we asked them whether they used our prototype to explore any of our use cases. If they did, we provided two 6-point Likert scales (strongly disagree – strongly agree) for each use case: (Q_1) whether VisFork provides the data needed to fulfill the use case and (Q_2) whether the user considers VisFork easier to use than the GitHub Network Graph (as a known baseline). Then, we asked each user to rate VisFork in general (5-star scale), grade the individual visualizations (5-point Likert scales, poor – excellent), and indicate whether they would recommend the tool to others (0 - not at all likely – 1 - extremely likely). Finally, any user could provide additional feedback and specify whether they would like to receive the results of our work. Please note that the GitHub users could skip any use case category in the survey if it did not apply to their typical work, which is why the numbers in our results do not always add up to 100 %.

Results. We received feedback from 10 GitHub community members, which is a response rate of approximately 11%. Most (80%) participants have between 10 to 20 years of experience with version control systems and more than five years of experience using forking (70%). Moreover, the participants indicated a strong interest in discovering the vitality and maintenance status of forks within ecosystems, with a particular focus on identifying those that are actively maintained versus transient or once-off forks. Other interests involve understanding the nature of changes, distinguishing bot activities, and seeing how as well as why forks evolve. Their extensive experience and interest in understanding properties related to our use cases underpin that the GitHub community members are well-acquainted to provide valuable feedback on VisFork. On the left side of Fig. 7, we display the participants' responses on the utility of VisFork (Q_1) and how it compares to the GitHub Network Graph (Q_2) for each use case (UC_i). So, each bar represents a statement about VisFork, and the segments in each bar denote the proportion of the participants' agreement.

UC_1: Exploring fork histories and relationships. Eight participants used VisFork to explore fork histories and relationships, leading to somewhat mixed views on the capabilities of our visualizations. When assessing the tool's ability to provide the necessary data to explore fork histories and relationships, five of the eight participants agreed with its usefulness, but the rest expressed (strong) disagreements. In contrast, more (six) participants acknowledged that VisFork is easier to use than GitHub's Network Graph. The negative feedback mostly focuses on implementation details, such as a lack of features to show or hide specific forks.

UC_2: Discover forks' states. Six participants used VisFork to explore forks' states and four gave positive ratings for both the tool's ability to provide the necessary data and its usefulness compared to GitHub's Network Graph. One participant, while giving moderately positive ratings, expressed a desire for more details in the visualization to shed light into which forks are actively merging changes from the original repository (i.e., keeping their forks up-to-date) and which are not. Among the two disagreements, one encountered technical issues with VisFork; the other found it impossible to explore forks' states in their own repository. We suspect that when the users query fork data ordered by stargazers (the default), the fetched forks (those with most stars) may not be actively maintained, leading to limited data. By enabling an advanced query and fetching the data ordered by date, this can be mitigated.

UC_3: Identify work in forks. Six participants used VisFork to identify work in forks. Four of them agreed that our prototype was able to provide the necessary data and five preferred it over the GitHub Network Graph. The word cloud appears to be a focal point for improvements. One participant mentioned that the word cloud does not effectively summarize major contributions, and another said it was not helpful, suggesting that the word cloud contained too much noisy information.

Summary. Overall, the feedback of the GitHub community members for our prototype was positive, with certain features being appreciated, especially the commit history. However, the participants also raised criticism regarding VisFork's functionalities in specific scenarios, such as the word cloud and classification based on commit messages. Some other concerns were mainly related to the layout and performance of the prototype, which are issues for engineering improvements. Nevertheless, the visualizations themselves seem to be effective based on the participants' feedback. The feedback also underpins the importance of a user-centric design and iterative testing. Addressing the highlighted concerns and continuously engaging with the user community can guide future enhancements to our work, but it seems to already make a valuable contribution for GitHub users keen on understanding fork ecosystems.

B. Student Participants

To strengthen our evaluation, we conducted a second user study for which we recruited three master students (invitation sent to

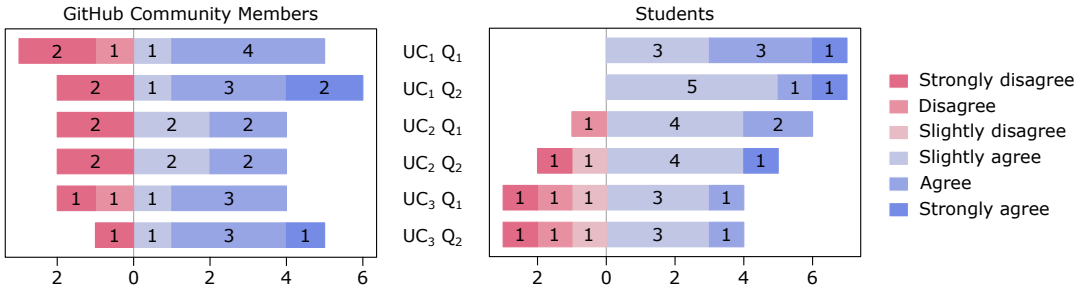

Fig. 7. Survey responses on VisFork from GitHub community members (left) and students (right). Note that the GitHub community members could decide not to answer a question, which is why the sums are not equal for all questions.

a course not connected to any author) and four PhD ones (not working with any author) in the field of software engineering. We asked them to complete the following tasks (matching our use cases), in this order, on our sample fork ecosystem; once using our prototype and once using the GitHub Network Graph:

UC$_1$ For the example repository, how many active forks (commits submitted actively) did you identify?

UC$_2$ The example repository "GreyAsteroid/open-in-mpv" was abandoned but some of the forks are still active (commits submitted actively). Please select any active forks you found.

UC$_3$ There are new features in the fork "xjbeta/iina-danmaku" but never merged back to the original repository. Could you name any new features you found?

We integrated these tasks into our previous survey (the one sent to the GitHub community members), which we further modified slightly for the students by removing some background questions.

Results. While all student participants had some experience with version control systems, two of them did not have any experience with using forks. For this reason, we first introduced them to the topic of fork ecosystems. On the right side of Fig. 7, we display the survey responses on the usability of VisFork and how it compares to the GitHub Network Graph. Note that the ratings between both groups of participants can be compared, since we asked the same questions, but that the way in which each group engaged with the respective tasks differs slightly (i.e., open exploration versus concrete tasks).

UC$_1$: Exploring fork histories and relationships. The student participants were asked to identify the number of active forks (i.e., those to which developers still commit) for the example ecosystem. Five of them provided a reasonable answer, while one misunderstood the question and simply counted recent commits. All participants agree that VisFork provides the necessary data and slightly agree that the prototype facilitates identifying active forks compared to the GitHub Network Graph. One student pointed out that our prototype offers better navigation, while GitHub's Network Graph has a more intuitive layout.

UC$_2$: Discover forks' states. Second, we asked the student participants to select all active forks of the example repository. Based on the different definitions of an active fork, the participants gave various answers, but some forks were frequently selected. So despite small differences, there seems to be a common understanding among the students about what constitutes an active fork. The frequently selected forks exhibit clear indicators of activity, such as recent commits or pull requests. Six participants were positive about our prototype, and five preferred it over GitHub's Network Graph.

UC$_3$: Identify work in forks. Lastly, we asked the student participants to name any new features in a specific fork that have not been merged back to the original repository. Some participants started by checking the commit classification and identified that the fork was an adaptive version of the original repository. When identifying the new features, the participants were able to extract at least one feature from the word cloud. However, their perception about VisFork helping identify hidden work in forks is more or less split.

Summary. Those student participants less familiar with fork ecosystems sometimes had difficulty understanding certain visualizations. Consequently, there is a need to enhance the user guidance of our visualizations using instructions and detailed legends. Despite some implementation-related criticisms, the visualizations of VisFork were well-received, showing their potential to offer insights into the dynamics and relationships of forks. In particular, we noted that the student participants used every visualization for the use cases for which we considered them most valuable.

VI. Discussion

Next, we discuss the effectiveness of our visualizations, our research objectives, and threats to the validity of our work.

A. Effectiveness

We evaluated the effectiveness of our prototype, VisFork, by eliciting feedback from two groups of participants: GitHub community members and student participants. The GitHub community members had extensive experience with the GitHub Network Graph, ensuring that they were familiar with forking. Across both groups, most participants found VisFork to be effective in providing the necessary data to explore fork histories and relationships for the different use cases. They particularly appreciated the commit history, which allowed them to gain insights into the temporal dynamics of a fork ecosystem. In terms of understanding forks' states, VisFork was generally viewed positively, helping users locate active forks based on criteria like recent commits. Some participants desired deeper insights into forks that regularly merge changes from the original repository. For identifying work in forks, the commit classification was beneficial in discerning different types of

commits. However, the word cloud feature got mixed feedback, due to its perceived noisiness in conveying key contributions.

The negative feedback for our prototype connects primarily to its implementation, indicating future improvements. Despite such criticism, we note that the underlying visualizations have been well-received and effective. So we argue that, while engineering challenges remain, our work contributes valuable insights on what visualizations are useful, and can be expanded in future research. For instance, the commit history allows users to gain insights into the temporal dynamics of fork ecosystems. Similarly, the commit classification proved useful in identifying different types of commits, offering valuable information about the nature of changes in a fork. An improved keyword scraping algorithm, such as analyzing commit messages along with code changes instead of solely relying on commit messages [23], [24], may improve the word cloud.

The student participants engaged with VisFork and the GitHub Network Graph to complete specific tasks related to fork exploration. While some of them were able to successfully identify active forks and extract features from the word cloud, others faced challenges in understanding the exact task requirements. We observed that users with less experience with fork ecosystems sometimes found it difficult to understand some visualizations. Interestingly, the challenges increased when investigating forks in more and more detail (i.e., UC_3), implying that there is a need for even better and simpler visualizations. In fact, most participants highlighted potential improvements of the implementation, but reinforced the validity and usefulness of the visualizations in VisFork, especially when compared to the GitHub Network Graph. By improving technicalities and refining the visualizations, VisFork has the potential to become a powerful and indispensable tool for GitHub users seeking deeper insights into fork ecosystems.

B. Research Objectives

To help developers understand the evolution of fork ecosystems ($\mathbf{RO_1}$), detailed fork histories and relationships should be visualized, indicating the various states of forks. Visualizing commit histories is vital, as it provides insights into timelines and relationships between commits. Furthermore, a history of collaboration networks can illuminate patterns between contributors and forks. It is crucial to have mechanisms for classifying commits, potentially utilizing advanced models, as well as multi-dimensional visual aids like Sankey diagrams and word clouds to facilitate pattern recognition and trend analysis for commit messages and activities.

As we discussed in Sec. IV, VisFork provides an overview of fork histories, relationships, and states ($\mathbf{RO_2}$). The commit history offers a detailed look into the chronological sequence of commits, showing the relationships between them and helping to track the evolution of forks over time. Meanwhile, the collaboration network history highlights the dynamic interactions between contributors and forks. In addition, multi-dimensional visual aids like Sankey diagrams allow to effectively categorize commits, while word clouds summarize and highlight patterns in commit messages. This, in turn, allows users to identify variations and unique characteristics within different forks.

The results of our user studies indicate that VisFork provides more detailed and interactive visualizations than the existing official tool, GitHub's Network Graph ($\mathbf{RO_3}$). GitHub users and students appreciated VisFork's commit history visualization for tracking fork evolution, and its collaboration network history for understanding interactions. While there were areas for improvement, especially in the implementation of specific features, the underlying visualizations of VisFork were largely seen as effective in providing insight into fork ecosystems. Most importantly, both groups were positive about our visualizations, had comparable agreement regarding their feasibility for our use cases, welcomed our work, and had many suggestions for future work.

C. Threats to Validity

The external validity of our work is threatened by the limited number of participants in our study. However, we involved practitioners interested in the topic as experts (GitHub community members) and aimed to further mitigate this threat by involving independent students—with both groups contributing similar feedback. Also, the external validity is threatened since we provided comparably little example data for our study. This data does not capture the full complexity of real-world fork ecosystems and does not allow us to reason on our work's scalability. These properties were not within the scope of this paper, but we aimed to mitigate them by allowing the GitHub community members to explore their own projects, with some of them indicating to have done so.

Lastly, we acknowledge that the design and perceived usefulness of any visualization depends on personal preferences and characteristics (e.g., color blindness). So, the internal validity of our work is threatened since we elicited opinions only and did not conduct a controlled experiment to collect quantifiable data. However, conducting controlled experiments on visualizations is very challenging and remains prone to subjective perceptions. We argue that our user study with different populations represents a feasible approach to understand how the visualizations can help developers in practice.

VII. Conclusion

In this paper, we presented visualizations for gaining insights into fork ecosystems, which we implemented in our prototype VisFork. To evaluate these visualizations, we conducted user studies with GitHub community members and graduate students. The user studies demonstrated that the visualizations can effectively support users in understanding fork ecosystems. Still, we also identified areas for improvement, particularly regarding technicalities and information cleansing. Overall, VisFork is a promising advancement on visualizing and comprehending the relationships within fork ecosystems. By integrating the user feedback, VisFork can become a valuable resource for developers and researchers who wish to gain a thorough understanding of fork histories, relationships, and work patterns. For this purpose, we plan to contact GitHub to see whether we can integrate and expand our work into the platform itself.

References

[1] M. Barnett, C. Bird, J. Brunet, and S. K. Lahiri, "Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets," in *International Conference on Software Engineering (ICSE)*. IEEE, 2015, pp. 134–144.

[2] M. Biazzini and B. Baudry, ""May the Fork be with You": Novel Metrics to Analyze Collaboration on GitHub," in *International Workshop on Emerging Trends in Software Metrics (WETSoM)*. ACM, 2014, pp. 37–43.

[3] S. Chen, L. Cleophas, and J. Krüger, "A Comparison of Visualization Concepts and Tools for Variant-Rich System Engineering," in *International Systems and Software Product Line Conference (SPLC)*. ACM, 2023, pp. 153—-159.

[4] L. Choshen and I. Amit, "ComSum: Commit Messages Summarization and Meaning Preservation," *CoRR*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2108.10763

[5] A. N. Duc, A. Mockus, R. Hackbarth, and J. Palframan, "Forking and Coordination in Multi-Platform Development," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2014, pp. 59:1–10.

[6] S. Duszynski, "Visualizing and Analyzing Software Variability with Bar Diagrams and Occurrence Matrices," in *International Software Product Line Conference (SPLC)*. Springer, 2010, pp. 481–485.

[7] S. Duszynski, V. L. Tenev, and M. Becker, "N-way Diff: Set-Based Comparison of Software Variants," in *Working Conference on Software Visualization (VISSOFT)*. IEEE, 2020, pp. 72–83.

[8] N. A. Ernst, S. M. Easterbrook, and J. Mylopoulos, "Code Forking in Open-Source Software: A Requirements Perspective," *CoRR*, 2010. [Online]. Available: https://doi.org/10.48550/arXiv.1004.2889

[9] S. Gharbi, M. W. Mkaouer, I. Jenhani, and M. B. Messaoud, "On the Classification of Software Change Messages Using Multi-Label Active Learning," in *Symposium on Applied Computing (SAC)*. ACM, 2019, pp. 1760–1767.

[10] G. Gousios, M. Pinzger, and A. van Deursen, "An Exploratory Study of the Pull-Based Software Development Model," in *International Conference on Software Engineering (ICSE)*. ACM, 2014, pp. 345–355.

[11] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective," in *International Conference on Software Engineering (ICSE)*. ACM, 2016, pp. 285–296.

[12] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective," in *International Conference on Software Engineering (ICSE)*. IEEE, 2015, pp. 358–368.

[13] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic Classification of Large Changes into Maintenance Categories," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2009, pp. 30–39.

[14] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang, "Why and How Developers Fork What from Whom in GitHub," *Empirical Software Engineering*, vol. 22, no. 1, pp. 547–578, 2017.

[15] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The Promises and Perils of Mining GitHub," in *International Working Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 92–101.

[16] S. Krieter, J. Krüger, T. Leich, and G. Saake, "VariantInc: Automatically Pruning and Integrating Versioned Software Variants," in *International Systems and Software Product Line Conference (SPLC)*. ACM, 2023, pp. 129–140.

[17] J. Krüger, "Understanding the Re-Engineering of Variant-Rich Systems: An Empirical Work on Economics, Knowledge, Traceability, and Practices," Ph.D. dissertation, Otto-von-Guericke University Magdeburg, 2021.

[18] J. Krüger and T. Berger, "An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2020, pp. 432–444.

[19] J. Krüger, Y. Li, C. Zhu, M. Chechik, T. Berger, and J. Rubin, "A Vision on Intentions in Software Engineering," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2023, pp. 2117–2121.

[20] J. Krüger, A. Mikulinski, S. Schulze, T. Leich, and G. Saake, "DSDGen: Extracting Documentation to Comprehend Fork Merges," in *International Systems and Software Product Line Conference (SPLC)*. ACM, 2023, pp. 47–56.

[21] J. Krüger, M. Mukelabai, W. Gu, H. Shen, R. Hebig, and T. Berger, "Where is My Feature and What is it About? A Case Study on Recovering Feature Facets," *Journal of Systems and Software*, vol. 152, pp. 239–253, 2019.

[22] M. Krzywinski, J. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra, "Circos: An Information Aesthetic for Comparative Genomics," *Genome Research*, vol. 19, no. 9, pp. 1639–1645, 2009.

[23] J. Y. D. Lee and H. L. Chieu, "Co-Training for Commit Classification," in *Workshop on Noisy User-Generated Text (W-NUT)*. ACL, 2021, pp. 389–395.

[24] S. Levin and A. Yehudai, "Boosting Automatic Commit Classification into Maintenance Activities by Utilizing Source Code Changes," in *International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*. ACM, 2017, pp. 97–106.

[25] R. E. Lopez-Herrejon, S. Illescas, and A. Egyed, "Visualization for Software Product Lines: A Systematic Mapping Study," in *Working Conference on Software Visualization (VISSOFT)*. IEEE, 2016, pp. 26–35.

[26] A. Mauczka, F. Brosch, C. Schanes, and T. Grechenig, "Dataset of Developer-Labeled Commit Messages," in *International Working Conference on Mining Software Repositories (MSR)*. IEEE, 2015, pp. 490–493.

[27] R. Medeiros, J. Martinez, O. Díaz, and J.-R. Falleri, "Visualizations for the Evolution of Variant-Rich Systems: A Systematic Mapping Study," *Information and Software Technology*, 2022.

[28] A. Mockus and L. G. Votta, "Identifying Reasons for Software Changes Using Historic Databases," in *International Conference on Software Maintenance (ICSM)*. IEEE, 2000, pp. 120–130.

[29] M. Mukelabai, C. Derks, J. Krüger, and T. Berger, "To share, or not to share: Exploring test-case reusability in fork ecosystems," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2023.

[30] L. Ren, S. Zhou, and C. Kästner, "Forks Insight: Providing an Overview of GitHub Forks," in *International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. ACM, 2018, pp. 179–180.

[31] L. Ren, S. Zhou, C. Kästner, and A. Wąsowski, "Identifying Redundancies in Fork-Based Development," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 230–241.

[32] M. U. Sarwar, S. Zafar, M. W. Mkaouer, G. S. Walia, and M. Z. Malik, "Multi-Label Classification of Commit Messages Using Transfer Learning," in *International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020, pp. 37–42.

[33] S. Schulze, J. Krüger, and J. Wünsche, "Towards Developer Support for Merging Forked Test Cases," in *International Systems and Software Product Line Conference (SPLC)*. ACM, 2022, pp. 131–141.

[34] D. Strüber, M. Mukelabai, J. Krüger, S. Fischer, L. Linsbauer, J. Martinez, and T. Berger, "Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems," in *International Systems and Software Product Line Conference (SPLC)*. ACM, 2019, pp. 177–188.

[35] Ş. Stănciulescu, S. Schulze, and A. Wąsowski, "Forked and Integrated Variants in an Open-Source Firmware Project," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 151–160.

[36] E. B. Swanson, "The Dimensions of Maintenance," in *International Conference on Software Engineering (ICSE)*. IEEE, 1976, pp. 492–497.

[37] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network Structure of Social Coding in GitHub," in *European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 2013, pp. 323–326.

[38] M. Yan, Y. Fu, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "Automatically Classifying Software Changes via Discriminative Topic Model: Supporting Multi-Category and Cross-Project," *Journal of Systems and Software*, vol. 113, pp. 296–308, 2016.

[39] S. Zhou, "Improving Collaboration Efficiency in Fork-Based Development," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1218–1221.

[40] S. Zhou, Ş. Stănciulescu, O. Leßenich, Y. Xiong, A. Wąsowski, and C. Kästner, "Identifying Features in Forks," in *International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 106–116.

[41] S. Zhou, B. Vasilescu, and C. Kästner, "What the Fork: A Study of Inefficient and Efficient Forking Practices in Social Coding," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2019, pp. 350–361.

[42] ——, "How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub," in *International Conference on Software Engineering (ICSE)*. ACM, 2020, pp. 445–456.