

# Hidden in the Code: Visualizing True Developer Identities

Stefano Campanella\*, Michele Lanza\*

\*REVEAL @ Software Institute – USI, Lugano, Switzerland

Email: [stefano.campanella, michele.lanza]@usi.ch

**Abstract**—Analyzing developers’ activity is an essential step in understanding a software project. When assessing the status of a project and studying its history and development, it is vital to assess the performance of each contributor with high precision. Unfortunately, as Version Control Systems (VCS) do not manage the concept of developer identity in a precise fashion, it is often hard to connect a single identity with all the contributions a user has made to the project, by taking into account potential aliases of the same person.

Our work aims to bridge this gap by processing the information related to the identities of contributors in the most popular projects available on GitHub, and to visualize their activity over time. The visualized identities are built on top of a disambiguation algorithm that aims to merge aliases of the same person into a single entity. Moreover, we perform bot detection to differentiate the activity made by bots from the one of humans. At last, we cluster the resulting identities to find users with the same activity patterns. By interacting with the proposed visualizations, one can precisely analyze the contributions of the people working on a project. Video URL: <https://youtu.be/O98IsBDBXKY>

**Index Terms**—Developers, Activity, Alias, Bots

## I. INTRODUCTION

Version Control Systems (VCS) store a variety of data including repository metadata, commit snapshots with author information and changes, branches and tags for managing project history, file contents, an index for staging changes before committing, metadata such as file permissions and timestamps, etc.

Previous studies have used VCS data to perform tasks on software repositories, such as gaining insights on projects and communities [1], [2] and performing bug prediction [3], despite known shortcomings, such as the fact that the history may be altered by the repository owner [4]. In recent studies, VCS data has also been the main source of information when analyzing developers’ activities [5]–[9], including their productivity [10]–[12].

To support the process of analyzing VCS data, researchers began to support the community by presenting various tools that aimed to enable better understanding and allow interaction with the data itself using visualizations. Often these tools aim for a broader approach, by combining information derived from different sources (e.g., mailing lists, issue trackers) or different projects of the same ecosystem (e.g., Gnome, Apache). Providing such a high level of interaction with the data often results in these tools not being able to address problems in a more fine-grained fashion.

Researchers looking into this data to investigate developers, often base their findings on the identities who are typically represented in VCS as commit signatures (full names and email addresses) of the committers and authors of the commit. However, it may happen that project members use more than one identity on the same repository, and by doing so spread their contribution in what for the system become different and independent identities. Overlooking this behavior may undervalue individual contributions, which can skew research results and have an impact on professionals using different collaborative development platforms.

Moreover, as bots are increasingly utilized in software development to streamline processes, improve efficiency, and enhance collaboration [13], it is important to consider them when employing and visualizing data regarding developers’ identities. Differentiating between automated and non-automated activities is essential to better understand the data gathered from VCS.

The focus of our work is to provide a comprehensive visualization of the community of contributors involved in the development of a specific software system that uses Git as version control system. By accurately and clearly representing the contributors, we aim to obtain a deeper understanding of the dynamics, structure, and behavior of the communities structure. To achieve this, we also address the long-standing challenge of aliasing, where a single developer may use multiple identities, and the presence of bots, which can be mistaken for human contributors. By resolving these issues, we ensure a more structured and well-defined representation of the community itself. To accomplish this, we employ a set of custom algorithms, built upon the state-of-the-art tools currently available. Furthermore, based on the contribution activity collected using the commits, we process the obtained identities to create clusters of users that behaved similarly during the contribution to the project.

By tackling these issues, we present a custom interactive visualization approach which provides a clearer comprehension of team dynamics and individual contributions, thereby facilitating more effective project management and collaboration [14]. Additionally, our visualization provides a more concrete understanding of the activity in the project over time, revealing trends, patterns, and insights of the project’s evolution, and enables the identification of key contributors and their levels of engagement, allowing for more effective collaboration and knowledge sharing.

## II. RELATED WORK

In the context of visualization, researchers have developed various interactive tools to visualize metadata in Git, providing a better understanding of large software development projects through visual representation. Dominos, developed by Da Silva *et al.* [15], explores project relationships, including connections between developers, classes, methods, and other collaborators. Similarly, Githru, presented by Kim *et al.* [16], offers a visual analytics system that allows users to filter metadata by specific authors and timeframes. Furthermore, Git-Truck, developed by Højelse *et al.* [17], provides visualizations of contributors and enables the manual merging of aliases to analyze their histories as a single entity. Stephany *et al.* proposed the first tool aimed at analyzing and visualizing OSS developer communities named Maispion [18]. The tool leverages mailing lists and version repositories to generate visualizations and provides insights into the ecosystem of OSS development. Subsequently Dueñas *et al.* introduced Perceval, which is able to perform automatic data gathering from tools related with contributing to open source development, *e.g.*, source code management, and issue tracking systems while keeping into consideration the existence of aliases belonging to the same individual [19]. Feist *et al.* presented TypeV, which leverages abstract syntax trees to show the activity of users, while also allowing to manually merge identities [20]. Lastly, Moreno *et al.* proposed SortingHat, a tool that helps to track and manage unique identities of project members in software projects, particularly open-source ones. It merges and matches contributor identities across different platforms, such as email and username, and provides additional information on members' gender, country, and organization affiliations. The tool allows for interactive manipulation of identities and bulk loading via batch files, making it a useful solution for projects with large communities [21]. Whilst these tools provide good insights, they depend on the final users that are required to manually add relationships between accounts. Moreover, most of these solutions are only useful when applied from the start of the project lifecycle, thus making them not easily applicable to already existing projects.

Identity disambiguation has been extensively researched across multiple domains. In the field of software engineering, researchers have proposed a variety of approaches to undertake this challenge, drawing on machine learning, natural language processing, and data mining. Furthermore, as Robles *et al.* [22] discovered, integrating data from heterogeneous sources makes identity disambiguation even more difficult [23], [24]. Existing identity disambiguation algorithms can be broadly classified into two categories: Endogenous and exogenous approaches [25]. Endogenous approaches operate within a “closed world” assumption, relying solely on data from the original repositories to disambiguate identities based on string similarities [26]. In contrast, exogenous approaches leverage external information, such as mailing lists, to resolve ambiguities [27]. Over the years, various algorithms have been proposed to address this issue [28], [29].

Bird *et al.* [30] pioneered the use of string similarity to identify correlations between aliases, demonstrating the feasibility of relation detection based on basic attributes like emails and usernames. Subsequent studies built upon this foundation, either enhancing performance through refined techniques [31] or introducing novel heuristics [25], [32], [33]. Comparative analyses of various approaches [34], [35] revealed that while existing algorithms can effectively address the problem to some extent, there remains room for improvement. Notably, Gambit [31] currently stands as the most effective endogenous approach, achieving a disambiguation rate of 98% within a repository through a series of heuristics evaluated against a manually constructed ground-truth. Furthermore, Amreen *et al.* [33] proposed ALFAA, an active learning-based disambiguator that leverages commit messages, edited files, name, and email information to improve identity resolution.

Recent studies have highlighted the significant impact of bots on various aspects of software development, including automated code reviews, continuous integration and deployment, and issue management [36]–[39]. For instance, bots like Dependabot and Renovate have become crucial in managing dependencies and maintaining security by automatically updating libraries and frameworks [40]. Identifying bot accounts is challenging, which complicates socio-technical analysis that requires differentiating between human and bot behavior [41]. Given the now known impact, researchers have started to find solutions to identify the presence of bots within developer communities, particularly in the context of OSS.

Dey *et al.* proposed BIMAN [13], a systematic approach to detect bots using author names, commit messages, files modified by the commit, and projects associated with the commits achieving an AUC-ROC of 0.9. Later, Golzadeh *et al.* tried to detect the presence of bots in distributed software development activity by leveraging the pull requests and issues comments of GitHub accounts [42], [43] resulting in a tool named BoDeGHa. Based on the ground-truth dataset proposed on the paper, the authors were able to achieve a very high weighted average precision, recall and F1-score of 0.98 on a test set containing 40% of the data. Moreover, they applied the same techniques on commit messages [44] with their tool named BoDeGic, achieving a precision of 0.77 on the dataset proposed by Dey *et al.* [13]. Along the same lines, Abdellatif *et al.* [45] proposed BotHunter, a machine learning-based approach to identify the bot accounts regardless of their activity level by selecting and extracting 19 features related to the account's profile activities. The model evaluation showed an F1-score of 92.4% and AUC of 98%. Lastly, Chidambaram *et al.* introduced Rabbit [46], an open source command-line tool that queries the GitHub Events API to retrieve the recent events of a given GitHub account and predicts whether the account is a human or a bot. The tool, which aims to reduce the GitHub API notably high consumption of other bot detection tools, achieves an AUC, F1 score, precision and recall of 0.92.

Our work aims to leverage state-of-the-art techniques to achieve a concrete and reliable representation of the users involved in the development of software systems.

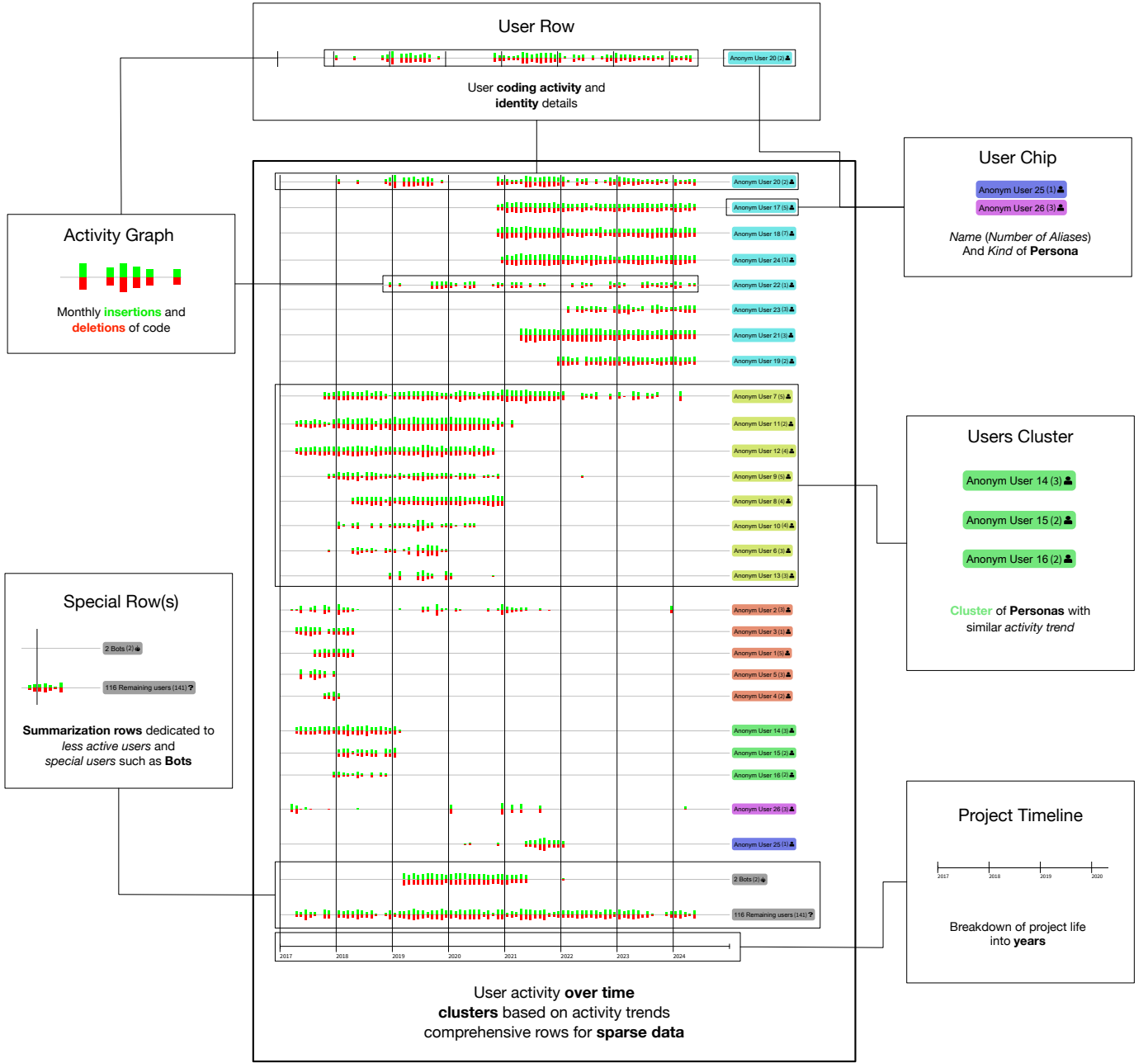


Fig. 1: Detailing the visualization approach

### III. VISUALIZATION

The goal of our visualization is to foster the exploration of the developers involved in a software system. Some pre-processing steps are needed to better represent our domain of interest: *Individuals*.

Figure 1 shows an example of our visualization on an average-sized repository *i.e.*, Webui 🐙. The project counts 139 unique GitHub Contributors and 144 unique Identities disambiguated with our own algorithm described in Section IV. As of June 12 2024, the repository counts 24,383 commits over more than seven years of life.

#### A. User Rows

The *User Row* is the heart of our visualization. Its goal is to represent the *Persona* (definition in Section IV) and their activity in the repository, in terms of added and removed lines of code. The *User Row* also serves as a representation element for the *Special Rows* that the visualization may include at the bottom, as we will discuss later. Figure 2 shows an example of *User Row*, where (A) represents the *Activity Graph Timeline* and (B) represents the *User Chip*.

**Activity Graph Timeline.** It aims to visualize the activity of the user in the repository over time. As shown on the left side of Figure 2, it is composed of three main elements:



Fig. 2: Detailed example of User Row

- — **Timeline.** This line, aligned with the timeline at the bottom of the visualization, helps the user to keep track of the position of the activity in time. The vertical elements correspond to the start of a new year.
- **Insertions Blocks.** The green blocks represent the amount of insertions the user committed to the source code in a specific month; the higher the box, the more insertions there were.
- **Deletion Blocks.** The red blocks represent the amount of deletions the user committed to the source code in a specific month; the taller the box, the more insertions there were.

Each column of this activity timeline is composed of one insertion and one deletion block and refers to a month of activity in the lifetime of the repository, for the given Persona. To reduce graphical complexity, the height of the insertions and deletions blocks is computed by smoothing the exact number of insertions/deletions collected during the mining with a natural logarithm function.

**User Chip.** This UI component encapsulates information pertaining to an individual's identity. These chips serve as a concise and efficient way to display key identity details. An example can be found on the right side of Figure 2. The chips display four important pieces of information.

Firstly, the user can read the name inferred using the NER algorithm (for privacy purposes, in this study, the names have been anonymized). Then, between brackets, it is possible to read the number of aliases that hide behind the same Persona. On the rightmost part of the chip, an icon helps to distinguish the three possible types of Persona, based on the bot detection execution.

The three different types are:

- **Human.** This type is assigned to the Personas which are composed of aliases that are detected as humans by the bot detection tool.
- **Bot.** A Persona is assigned the Bot type when all the aliases linked to it are detected as bots.
- **Unknown.** When the bot detection process fails to predict the type of a Persona, or the aliases are both bots and humans, the system automatically assigns the unknown type. This type is also used for grouped elements with at the least two different types of Personas inside.

Lastly, the color of each chip corresponds to the cluster to which the individual belongs, providing a quick visual reference for group affiliation. By hovering on the chip, the user will see a list of information on the user such as the number of months of activity (Figure 4a).

## B. Clusters

The clusters of User Chips shown in the visualization (right side of Figure 1) are the representation of the results of a clustering algorithm, described in Section IV. The goal is to group the identities based on different activity patterns, *i.e.*, contributors working on the project in different years.

Each cluster is defined by a specific color which is used as background for all the chips of the Personas belonging to the same cluster. From top to bottom, the elements of each cluster are ordered from most to least active within the cluster itself. Clusters are also ordered from most to least active. The space between each cluster is designed to help the user make the distinction between clusters easier. This clear visual separation ensures that users can quickly and accurately identify the activity levels and groupings of contributors.

## C. Special Row(s) and Project Timeline

The bottom of the visualization, together with the Timeline that tracks the lifetime of the repository, is dedicated to all the optional *Special Row(s)*. Differently from simple rows that represent a single Identity, this rows aim to represent a specific group of contributors of the repository.

Given that a repository may have a very large number of contributors, it is sometimes necessary to reduce the amount of data by grouping the less active users (*Remaining Users*) on a single line. A *Special Row* named *Remaining Users* (Figure 3 (B)), is designed to do exactly that. For this reason, the visualization will always have such a row at the bottom of the graph. Moreover, when the list of contributors includes (detectable) bots, we employ an ad-hoc *Special Row* to group all the bots in a single activity line, as shown in Figure 3 (A). This group is especially useful to summarize the activity of bots in the repository and understand their overall impact on the project. Lastly, Figure 3 (C) highlights the Timeline of the project. In this timeline we track the years of activity by drawing a vertical line across the entire visualization, to represent the sequence and timing of events. Figure 1 summarizes the visualization, highlighting all the UI elements that compose it.



Fig. 3: Example of Special Rows

We also introduced a set of interactions with the visualization, *i.e.*, filtering, tooltips and interactive menus, that enable deeper exploration, allowing users to navigate the data in a more customized and accurate manner.

#### D. Interactions

As previously said, the volume and popularity of the project itself determine how many contributors a repository has. Usually (especially in OSS projects) the number of contributors is quite high and difficult to represent. For this reason, our visualization performs pre-processing before displaying the data. Nevertheless, sometimes preprocessing cannot reduce the noise in the data sufficiently to make the visualization usable. For this reason we provide to the user a list of filters, useful to further reduce the number of identities visualized at once.

Users can thus enable the filters and cut the visualized identities based on two different criteria: (a) The minimum number of commits the identity should have performed over the years, and (b) the number of months of activity in which the individual contributed to the project. By doing so, it is possible to reduce the list of relevant contributors and simplify the visualization.

Additionally, by right-clicking the User Chip (Figure 4b), one can check all the information regarding the Persona’s activity, including the list of commits that they made, and breakdown the Persona to display all the aliases on a freshly built visualization that adheres to the same criteria.

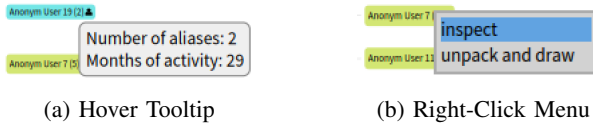


Fig. 4: User Chip Interactions

Figure 5 shows an example of the decomposed Persona visualization.

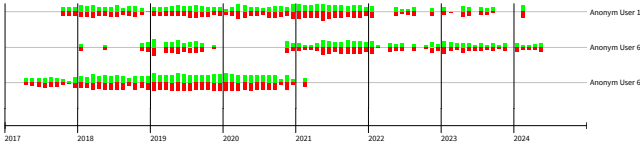


Fig. 5: Unpacked Persona visualization

#### IV. TECHNICAL BACKGROUND

Our goal is to visualize and interact with identities that contain all information generated by the same person, even if created under different aliases within the versioning system. For this reason, we introduce the concept of “Persona”, that represents the individual in the context of a repository. Moreover, we process the information collected for each individual to find the best way to identify them (*i.e.*, what name represents the Persona better among all the aliases), and to model and adapt the data to our goal of visualizing the activity in the repository over time. Figure 6 depicts all data processing techniques.

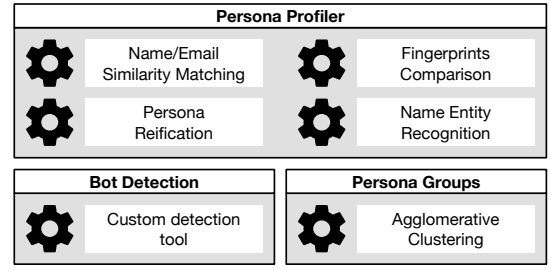


Fig. 6: Overview of the approach employed under the hood

#### A. Data Collection and Manipulation

To support our analysis and observations, we collect data from well-known OSS by relying on GitHub’s Stars system, which lets users bookmark and show appreciation for repositories. A repository’s popularity among other GitHub users can be roughly estimated by looking at its star count. High star count repositories are typically thought to be more dependable, useful, and popular. As a result, we sort all GitHub public repositories by star count and mined data from 7 of the top 10. Three projects were excluded as their size made data collection prohibitive for the purposes of our study. Table I shows the list of repositories taken into consideration.

TABLE I: Top 10 popular projects in GitHub (28 May 2024)

Name	Commits	Stars	Utilized
freecodecamp	35,430	389,886	Y
developer-roadmap	4,175	275,869	Y
react	18,716	223,064	Y
vue	3,590	207,037	Y
tensorflow	163,529	182,509	N
linux	1,275,250	171,548	N
ohmyzsh	7,235	169,475	Y
bootstrap	22,800	167,621	Y
flutter	40,795	162,153	Y
vscode	121,425	158,513	N

**Data Preprocessing.** Before being visualized, the data of the Personas collected from the repositories undergoes a series of pre-processing steps. We collect all the insertions and the deletions for each month of activity for each persona. Using months as the time granularity provides a number of benefits.

Computation time is reduced without information loss, while the overall visualization is still able to communicate the activity for each Persona and the overall activity of the repository under analysis. Moreover, the number of insertion-/deletions for each month are smoothed using a logarithmic function, to make the data more resilient to exceptional or recurrent short events, where the users are much more active than usual. Lastly, we detect the main contributors of each repository to avoid the visualization of the big slice of developers, that only contributed to the project with a handful of commits in a really short amount of time. To do so, we need to detect the most frequent committers. We compute the first ( $Q_1$ ) and third ( $Q_3$ ) quartile in terms of number of commits, and the interquartile range ( $iqr$ ) [47], [48].

As most of the contributors of the repository have low activity, the outliers over  $Q_3$  are the users with the most contributions. We detect *major contributors* using Tukey’s formula for outlier detection [49]. Thus we label as *major contributors* the contributors with  $n_{\text{commits}} > Q_3 + 1.5 \times iqr$  and aggregate all the others as *minor contributors*.

### B. Disambiguation

To merge aliases and disambiguate identities by only relying on data available in VCS such as Git, we implement a two-step algorithm that takes into consideration not only names and emails, but also other information available directly from the Git logs (such as the files edited in the contribution an Alias makes to the repository). The algorithm is composed of two main pillars: *string similarity*, and *fingerprint* heuristics. While the algorithm mainly relies on *string similarity*, the *fingerprint* heuristic also contributes significantly to the results of the disambiguation. Moreover, our technique is independent of external resources and can be applied to any repository with a Git history.

**String similarity.** We adopt the techniques applied in Gambit [31]. Gote *et al.* computed the similarity between different entities using a set of ten rules, which involve comparing full names and emails, as well as first, middle, and last names, or email bases directly extracted from the name and email strings. To gather such data for the comparisons, preprocessing is applied to extract elements such as parts of the names and email bases from the input strings. To reduce false positives, a similarity threshold is required, and based on Gambit’s authors’ guidelines, the optimal threshold is 0.95.

**Fingerprints.** We build our set of *fingerprints* on the features employed in ALFAA [33]. Specifically, we implement a comparison among the files edited by each alias by constructing a vector for each alias that indicates whether or not a file was edited (*e.g.*,  $[0, 1, 0]$  where  $F_0$  and  $F_2$  are not edited, and  $F_1$  is edited). The list of file is computed as the intersection of the files edited by the two aliases that are being compared. Then, we compute the cosine similarity. By doing so, we aim to catch cases in which the same developer edited the same group of files using different aliases.

As output of this process, we obtain a list of Personas. Each Persona gathers all the aliases detected as owned by the same individual, encapsulating the information regarding all the contributions the individual has made to the repository.

### C. Name Recognition

The goal of this step is to identify and select the most appropriate human name from a list of potential names, gathered from all the aliases merged together during the disambiguation of the identities. We do so by leveraging natural language processing techniques. First, we process each name to handle formatting inconsistencies, as names may be inconsistently formatted. Then, we rank them based on their likelihood of being recognized as person names from a Name Entity Recognition [50] model, and then select the highest-scoring name as the name of the Persona.

**Ethical Concerns.** For privacy reasons, despite being publicly available data, in the following study all the users involved in the analyzed repositories will be anonymized.

### D. Bot Detection

To perform bot detection at state-of-the-art level, we decided not to rely on a specific tool implementation, but to combine most of them to achieve higher performance, at the cost of computational time. Thus, we implement a custom algorithm that employs most of the current available bot detection tools in the research community. Specifically, both BoDeGHa [42] and BoDeGic [44] as well as Rabbit [46] are used as subroutines for our algorithm. On top of this three external tools, we add a more heuristic-based approach, inspired by BIN which is part of the BIMAN approach for bot detection [13]. The heuristics step of our detection aim to filter out the obvious bot accounts: Those containing strings such as *bot*, *test*, *auto*, *actions* either in their name or GitHub account username.

After running these four detection approaches on all the users in the repository under analysis, we compute a weighted average of the results to conclude whether the user is a bot or a human. The weighted average also keeps into consideration the fact that the three CLI tools employed in this process often are not able to assign any label to a given account, thus giving no results. In this case the heuristics comes in handy as it helps filtering a not so small set of users that otherwise would have gone unnoticed. The weights are applied as follows:

$$\begin{aligned} \text{Heuristics} &= 0.30 & \text{BoDeGHa} &= 0.25 \\ \text{BoDeGic} &= 0.20 & \text{Rabbit} &= 0.25 \end{aligned}$$

### E. Clustering

We perform clustering on the major contributors of the systems, with the goal of finding users with the same activity patterns over time in the lifespan of the repository. This helps to identify groups of people that joined forces for a specific amount of years in the development process, or can show how the evolution of the project led to a change of contributors activity. We employ Agglomerative Clustering on the distribution of activity of each Persona per month, computing an array as follows:

$$\mathbf{A} = [\lceil \ln(I_1 - D_1) \rceil, \lceil \ln(I_2 - D_2) \rceil, \dots, \lceil \ln(I_N - D_N) \rceil]$$

The array contains, for each month of the repository lifetime (N), the net value of insertions and deletions performed by the Persona. In our analysis we smooth the data with the natural logarithm to avoid exceptional values and normalize the values between different Personas. The arrays are then clustered using Agglomerative Clustering. This technique is a type of hierarchical clustering method used to group objects into clusters based on their similarity using linkage distance, which is the shortest distance between two points in each cluster. It follows a bottom-up approach, starting with each object in its own cluster and iteratively merging the closest pairs of clusters until a single cluster or a specified number of clusters is formed. This method does not require a fixed number of clusters as input as opposed to common algorithms such as K-means, ensuring more flexibility to our approach.

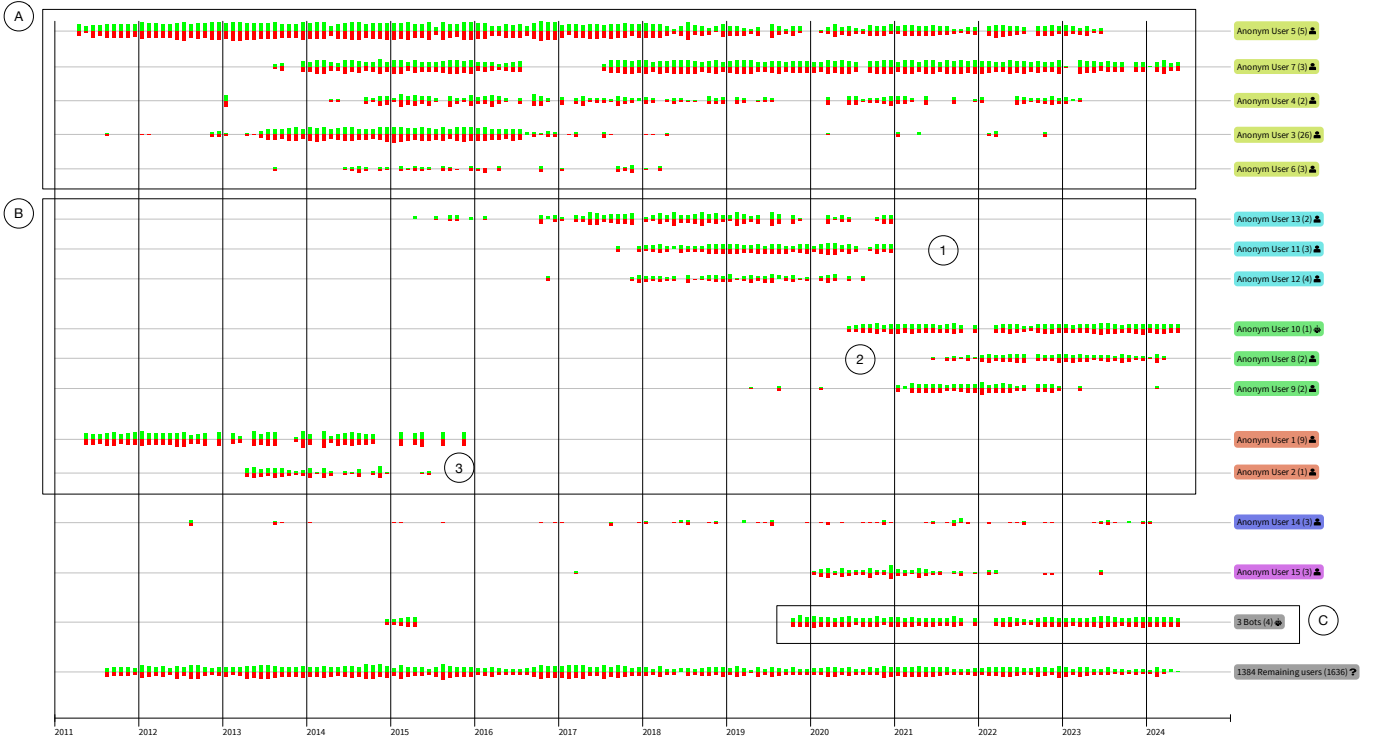


Fig. 7: Visualization applied to *Bootstrap* GitHub repository

## V. UTILITY AND PRACTICAL APPLICATIONS

To evaluate the effectiveness of our visualization, we illustrate our approach on three different projects. The visualizations we are going to analyze were obtained after applying some filters to reduce the complexity of the data. Table II displays the three repositories and the filters we applied on them.

TABLE II: Case-study Repositories and Filters

Repository	Filters applied:	
	Commits	Activity Months
React	35	20
Bootstrap	35	12
Vue	10	6

For example, 35 commits and 12 activity months means that we only depict user rows that have performed at least 35 commits and that were active during at least 12 months, while those with less commits and fewer months are aggregated in the Remaining Users row.

### A. *Bootstrap*

Bootstrap is a popular front-end framework for developing responsive and mobile-first websites and web applications. As of June 12 2024, the project counts more than 1,250 contributors, 22,816 commits and more than 13 years of life. It is the 8th most popular software system publicly available on GitHub, in terms of stars. Figure 7 shows our visualization applied on the project, with the filters described in Table II.

At a first glance at the visualization, there is a clear distinction between the users that contributed to the project for most of its lifetime, and the ones that contributed to it for a specific time period. Specifically, the cluster (A) is composed of the most active contributors overall, and includes the ones that kickstarted the project in 2011 together with what looks like the most active user on the project overall.

Moreover, Figure 7 (B) highlights three small clusters of users that highly impacted the project at some point during its lifetime. Cluster (B3) is composed of two users, using 10 aliases in total (!), that contributed to the project in the first five years. Cluster (B1) shows three contributors (9 aliases) that were active from 2015 to 2020. Lastly, cluster (B2) shows three users (5 aliases) that started their activity in 2020 and persist to this day in contributing to the project. Finally, Figure 7 (C) points out the usage of automated code committers starting from 2020.

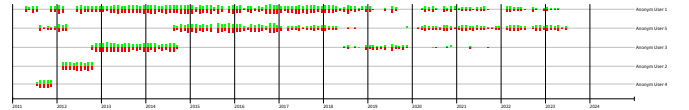


Fig. 8: Unpacked visualization for the first user in cluster A

Figure 8 displays the unpacked visualization for the first user in the cluster (A). We can see that, out of the 5 detected identities, the user committed consistently with two of them throughout the years.



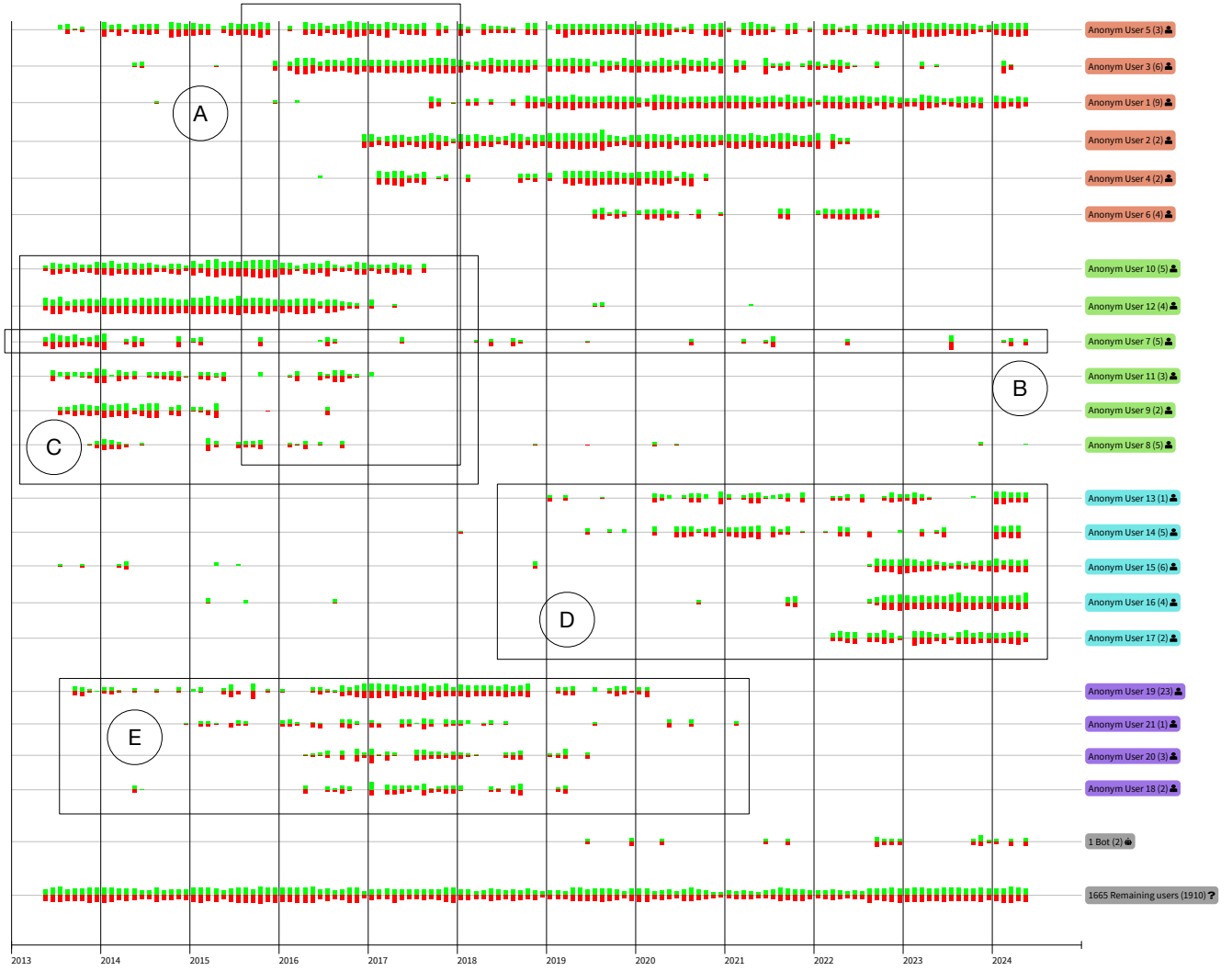


Fig. 9: Visualization applied to *React* GitHub repository

By analyzing Bootstrap with our visualization, we were able to gather some interesting insights on the contributors dynamics in the project. First of all, we could see how a group of developers remained active for the entire lifetime of the repository, with one of them being the most active contributor to the project overall. Moreover, we could see how other groups of developers contributed to the project for a smaller amount of time before stopping. This behavior was indeed needed to the project, as there always were at least two developers, external of the “main group” that actively contributed to the project. Lastly, we were able to see how at some point in time, it was decided to introduce bots as active code contributors.

On a side note, by checking out the usage of aliases, we can notice how most of the active users employed at the least two aliases when contributing to the project. In the case of more than two aliases, often most of them were employed for a really small contribution, while two of them carried most of the overall contribution of the User, as seen in Figure 8.

### B. *React*

React is a widely-used JavaScript library for building user interfaces. React has received contributions from over 1,500 people as of June 12, 2024, with more than 21,000 commits and nearly a decade of active development. It is the third most popular software project on GitHub in terms of stars, demonstrating its widespread adoption and community support. Figure 9 depicts our visualization of the React repository.

When analyzing the visualization, other than conclusions similar to the one described for Bootstrap in Section V-A, we can notice other trends. Figure 9 (A) highlights an interesting overlap of activity between the two most active clusters of contributors. This suggests a transition between contributors. In detail, we can notice how at the start of the project life *Anonym User 10* and *Anonym User 12* were two of the most active, if not the most active contributors of the repository, together with *Anonym User 5*. After 5 years of contributions, these two users started contributing less to the project until they eventually stopped.



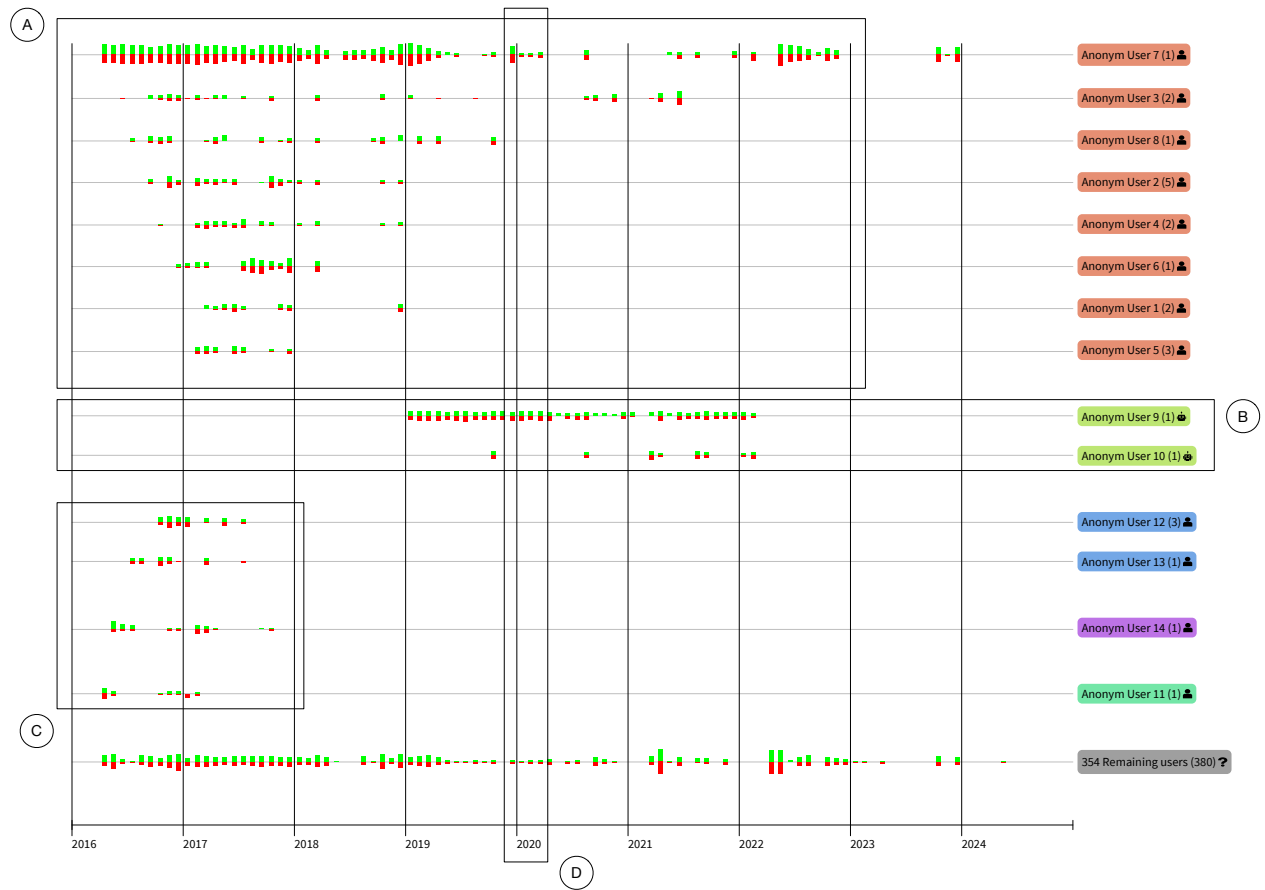


Fig. 10: Visualization applied to *Vue 2.0* GitHub repository

While they started decreasing in contribution, we can notice how new contributors started to be very active in the repository, i.e., *Anonym User 2*, *Anonym User 3* and *Anonym User 4*. This suggests that during this period there was a handover to new developers.

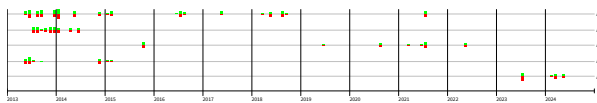


Fig. 11: Unpacked visualization for *Anonym User 7* (B)

Figure 9 (B), unpacked in Figure 11, highlights an odd behavior from one of the contributors. *Anonym User 7*, contributed sporadically over the entire life of the project, with the peak of activity being at the start of the project itself. Figure 9 (C), (D) and (E) highlights the “support” clusters. React is supported by a main group of contributors, but over the years other groups of developers heavily contributed to the project: Three distinct groups supported the main developers over the years. All of them for at most five years, and all of them overlapping in time. This suggests a handover in the sense of new groups led by someone that already had some knowledge about the project, starting to actively contribute to the project source code.

Regarding the use of aliases in the project, almost all developers contributed with more than one identity. Moreover, in the React project bots were not heavily employed.

### C. Vue

Vue is a JavaScript framework for building user interfaces. Vue has accumulated contributions from over 600 contributors, with more than 30,000 commits and over 9 years of development. The repository we analyze, *Vue 2.0*, is still under maintenance but has been superseded by version 3.0.

Figure 10 shows the results of our visualization on this project. Figure 10 (D) highlights the release year for *Vue 3.0*, which corresponds to the same year when the *Vue 2.0* project started to be under maintenance. We decided not to merge bots but to leave them as unfiltered users displayed in the visualization. Figure 10 (B) highlights a cluster that is exclusively populated with activity that is coming from bot users. The activity of the bots only lasted three years, and peculiarly the release year for *Vue 3.0* was right in between the activity of the two bots. Despite the short lifetime of the repository (less than five years of active development going from 2016 to 2020), the core of the implementation happened in the first two years of life, as Figure 10 (A) and (C) highlights.

After that, most of the development weight fell on the most active contributors (who are part of the most active cluster of contributors). It is also possible to notice how, after the release of the new version of the framework, the overall amount of activity decreased considerably, with a peak during 2022. While most of the activity is now dedicated to maintenance, the community still actively contributes to the project, as we can see from the “Remaining Users” row.

## VI. CONCLUSION

We presented a novel way to visualize the activity of the developers, using data from VCS such as Git. This visualization can greatly reduce the complexity of the data stored in version control systems, regarding both the activity and the identity of the contributors. We have applied our visualization on case studies to illustrate its usefulness.

### A. Reflection on the Results

The visualization is able to illustrate the intricate relationships and interactions among the community of developers in the context of software systems. The visualization offers a quantitative and spatial representation of these relationships, making it easier to identify central figures and peripheral participants in the community. It also highlights dense clusters of nodes representing developer collaborations and reveals patterns of interaction. It also support the investigation of activity around a single developer, by handling aliases in a precise fashion. Lastly, it helps to distinguish the kind of activity in the repository, by differentiating between humans and bots.

The visualization highlights who plays what role in the social structure of the system under analysis, and helps to easily identify who are the *Project Initiators* and *Core Members* in charge of guiding and coordinating the development of the software system, while also identifying the *Active* and *Peripheral Developers* involved in the project [51]. Moreover, it supports the observation of peculiar community dynamics such as takeovers in contribution [37]. Lastly, when analyzing systems that go through periodic update and release cycles, such as the Vue case study, it can support understanding on the dynamics of collaborations between developers in this key moments of the software system lifetime [52].

For community managers, the visualization could serve as a tool to identify key developers and potential mentors, facilitating targeted interventions to enhance community cohesion. For developers, it can provide a clearer picture of the social landscape, helping them navigate collaborations more effectively. Researchers can leverage the visualization to form new hypotheses about dynamics within community and identities, and further investigate the underlying social mechanisms.

### B. Limits of the Approach

Even with its usefulness, there are some limitations that the visualization inherits from the data pre-processing employed under the hood before the data gets visualized:

**Identity Disambiguation.** Although our disambiguation algorithm uses state-of-the-art approaches, identity disambiguation remains a critical problem. Our algorithm may still produce incorrect results, such as merging identities that do not belong to the same individual.

**Activity Clustering.** The clustering used in the visualization process offers a straightforward approach to group similar activity patterns. Its simplicity ensures efficient computational performance, but it may result in incorrect groupings within clusters. Additionally, the use of an algorithm that automatically generates an arbitrary number of clusters can lead to the creation of an excessive number of unnecessary clusters.

**Bot Detection.** The bot detection process is still unreliable. Specifically, most of the elements used for the bot detection itself rely on information gathered from GitHub, such as Issues. In some cases, there is not enough information for these tools to distinguish whether a user is a bot or not. Even if we tried to compensate this problem by combining tools and merging the result, this process can still yield erroneous results.

### C. Future Work and Improvements

By applying a more precise process of identity disambiguation and bot detection, we aim to enhance the reliability of the visualization. Moreover, we aim to improve the clustering algorithm and fine-tune its results. Lastly, to improve the resulting identities, we aim to improve the Name Detection algorithm, by employing a lightweight Large Language Model, that can capture more information from different aliases, *i.e.*, Real Name and Username, and merge them into a single string that could be used as identifier.

■ Video demonstration: <https://youtu.be/O98IsBDBXKY>

**Acknowledgments.** The authors would like to thank the Swiss Group for Original and Outside-the-box Software Engineering (CHOOSE) for sponsoring the trip to the conference and the Swiss National Science Foundation (SNF) for the financial support via the project “INSTINCT” (Project No. 190113).

## REFERENCES

- [1] H. C. Gall and M. Lanza, “Software evolution: Analysis and visualization,” in *International Conference on Software Engineering (ICSE)*. ACM, 2006, pp. 1055–1056.
- [2] S. Gong and H. Zhong, “Code authors hidden in file revision histories: An empirical study,” in *International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 71–82.
- [3] K. Muthukumaran, A. Choudhary, and N. B. Murthy, “Mining GitHub for novel change metrics to predict buggy files in software systems,” in *International Conference on Computational Intelligence and Networks (CINE)*. IEEE, 2015, pp. 15–20.
- [4] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, “The promises and perils of mining git,” in *International Working Conference on Mining Software Repositories (MSR)*. IEEE, 2009, pp. 1–10.
- [5] A. Mockus, R. T. Fielding, and J. D. Herbsleb, “Two case studies of open source software development: Apache and Mozilla,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

- [6] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *International Working Conference on Mining Software Repositories (MSR)*. ACM, 2008, pp. 129–132.
- [7] I. Scholtes, P. Mavrodiev, and F. Schweitzer, "From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in open source software projects," *Empirical Software Engineering*, vol. 21, no. 2, pp. 642–683, 2016.
- [8] R. M. Parizi, P. Spoletini, and A. Singh, "Measuring team members' contributions in software engineering projects using git-driven technology," in *Frontiers in Education Conference (FIE)*. IEEE, 2018, pp. 1–5.
- [9] S. Hamer, C. Quesada-López, A. Martínez, and M. Jenkins, "Measuring students' contributions in software development projects using git metrics," in *Latin American Computing Conference (CLEI)*. IEEE, 2020, pp. 531–540.
- [10] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *International Symposium on Foundations of Software Engineering (SIGSOFT)*. ACM, 2014, pp. 19–29.
- [11] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1178–1193, 2017.
- [12] E. Murphy-Hill, C. Jaspán, C. Sadowski, D. Shepherd, M. Phillips, C. Winter, A. Knight, E. Smith, and M. Jorde, "What predicts software developers' productivity?" *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 582–594, 2021.
- [13] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, and A. Mockus, "Detecting and characterizing bots that commit code," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2020, pp. 209–219.
- [14] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov, "The sky is not the limit: multitasking across github projects," in *International Conference on Software Engineering (ICSE)*. ACM, 2016, pp. 994–1005.
- [15] J. R. da Silva Junior, D. P. Campagna, E. Clua, A. Sarma, and L. Murta, "Dominoes: An interactive exploratory data analysis tool for software relationships," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 377–396, 2020.
- [16] Y. Kim, J. Kim, H. Jeon, Y. Kim, H. Song, B. H. Kim, and J. Seo, "Githru: Visual analytics for understanding software development history through git metadata analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 656–666, 2020.
- [17] K. Højelse, T. Kilbak, J. Røssum, E. Jäpelt, L. Merino, and M. Lungu, "Git-truck: Hierarchy-oriented visualization of git repository evolution," in *Working Conference on Software Visualization (VISOFT)*. IEEE, 2022, pp. 131–140.
- [18] F. Stephany, T. Mens, and T. Gırba, "Maispion: A tool for analysing and visualising open source software developer communities," in *International Workshop on Smalltalk Technologies (IWST)*. ACM, 2009, pp. 50–57.
- [19] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona, "Perceval: software project data at your will," in *International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. ACM, 2018, pp. 1–4.
- [20] M. D. Feist, E. A. Santos, I. Watts, and A. Hindle, "Visualizing project evolution through abstract syntax tree analysis," in *Working Conference on Software Visualization (VISOFT)*. IEEE Computer Society, 2016.
- [21] D. Moreno, S. Dueñas, V. Cosentino, M. A. Fernandez, A. Zerouali, G. Robles, and J. M. Gonzalez-Barahona, "Sortinghat: Wizardry on software project members," in *International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE / ACM, 2019, pp. 51–54.
- [22] G. Robles and J. M. González-Barahona, "Developer identification methods for integrated data from various sources," in *International Workshop on Mining Software Repositories (MSR)*. ACM, 2005.
- [23] B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of github teams," in *Working Conference on Mining Software Repositories (MSR)*. IEEE Computer Society, 2015, pp. 514–517.
- [24] T. Fry, T. Dey, A. Karnauch, and A. Mockus, "A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2020, pp. 518–522.
- [25] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "Who's who in gnome: Using LSA to merge software repository identities," in *International Conference on Software Maintenance (ICSM)*. IEEE Computer Society, 2012, pp. 592–595.
- [26] P. Christen, "A comparison of personal name matching: Techniques and practical issues," in *International Conference on Data Mining (ICDM)*. IEEE Computer Society, 2006, pp. 290–294.
- [27] Y. Xiong, Z. Meng, B. Shen, and W. Yin, "Developer identity linkage and behavior mining across github and stackoverflow," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 27, no. 9–10, pp. 1409–1426, 2017.
- [28] A. Eliseeva, Y. Sokolov, E. Bogomolov, Y. Golubev, D. Dig, and T. Bryksin, "From commit message generation to history-aware commit message completion," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 723–735.
- [29] W. Schueller, J. Wachs, V. D. P. Servedio, S. Thurner, and V. Loreto, "Evolving collaboration, dependencies, and use in the rust open source software ecosystem," *CoRR*, vol. 9, no. 1, p. 703, 2022.
- [30] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *International Workshop on Mining Software Repositories (MSR)*. ACM, 2006, pp. 137–143.
- [31] C. Gote and C. Zingg, "gambit - an open source name disambiguation tool for version control systems," in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 80–84.
- [32] J. Novak, P. Raghavan, and A. Tomkins, "Anti-aliasing on the web," in *International Conference on World Wide Web (WWW)*. ACM, 2004, pp. 30–39.
- [33] S. Amreen, A. Mockus, R. Zaretski, C. Bogart, and Y. Zhang, "ALFAA: active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1136–1167, 2020.
- [34] I. S. Wiese, J. T. da Silva, I. Steinmacher, C. Treude, and M. A. Gerosa, "Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, 2016, pp. 345–355.
- [35] M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Sci. Comput. Program.*, vol. 78, no. 8, pp. 971–986, 2013.
- [36] L. Erlenhov, F. de Oliveira Neto, A. Leitão, P. Leitner, and H. Grahm, "Current and future bots in software development," in *International Workshop on Bots in Software Engineering (BotSE@ICSE)*. IEEE / ACM, 2019, pp. 7–11.
- [37] S. Wagner and F. Deissenboeck, "Defining productivity in software engineering," pp. 29–38, 2019.
- [38] M. D. Storey, A. Serebrenik, C. P. Rosé, T. Zimmermann, and J. D. Herbsleb, "Botse: Bots in software engineering (dagstuhl seminar 19471)," *Dagstuhl Reports*, vol. 9, no. 11, pp. 84–96, 2019.
- [39] M. S. Wessel, M. A. Gerosa, and E. Shihab, "Software bots in software engineering: Benefits and challenges," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2022, pp. 724–725.
- [40] R. He, H. He, Y. Zhang, and M. Zhou, "Automating dependency updates in practice: An exploratory study on github dependabot," *Transactions on Software Engineering*, vol. 49, no. 8, pp. 4004–4022, 2023.
- [41] M. Golzadeh, A. Decan, and N. Chidambaram, "On the accuracy of bot detection techniques," in *International Workshop on Bots in Software Engineering (BotSE@ICSE)*, M. Wessel, Ed. ACM, 2022, pp. 1–5.
- [42] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in github issue and PR comments," *J. Syst. Softw.*, vol. 175, p. 110911, 2021.
- [43] N. Chidambaram and P. R. Mazrae, "Bot detection in github repositories," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2022, pp. 726–728.
- [44] M. Golzadeh, A. Decan, and T. Mens, "Evaluating a bot detection model on git commit messages," *CoRR*, vol. abs/2103.11779, 2021.
- [45] A. Abdellatif, M. S. Wessel, I. Steinmacher, M. A. Gerosa, and E. Shihab, "Bothunter: An approach to detect software bots in github," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2022, pp. 6–17.
- [46] N. Chidambaram, T. Mens, and A. Decan, "Rabbit: A tool for identifying bot accounts based on their recent github event history," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2024.
- [47] D. L. Whaley III, "The interquartile range: Theory and estimation," Master's thesis, East Tennessee State University, 2005.

- [48] P. D. Domanski, "Study on statistical outlier detection and labelling," *International Journal of Automation and Computing*, vol. 17, no. 6, pp. 788–811, 2020.
- [49] J. W. Tukey *et al.*, *Exploratory Data Analysis*. Pearson, 1977, vol. 2.
- [50] A. Mansouri, L. S. Affendey, and A. Mamat, "Named entity recognition approaches," *International Journal of Computer Science and Network Security*, vol. 8, no. 2, pp. 339–344, 2008.
- [51] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2003, pp. 419–429.
- [52] S. Panichella, G. Canfora, M. D. Penta, and R. Oliveto, "How the evolution of emerging collaborations relates to code changes: an empirical study," in *International Conference on Program Comprehension (ICPC)*. ACM, 2014, pp. 177–188.