Detecting Gender Representation Gaps in GitHub Projects: A Plugin for Core and Non-Core Role Analysis

Monica Daniella Homescu

June 26, 2025

Abstract

Despite open-source software (OSS) being associated with openness and inclusivity, gender representation still remains an issue within these communities. Women and gender minorities are significantly underrepresented, especially in influential roles. To bring attention to these discrepancies, we developed a plugin that analyzes GitHub repositories to identify gender representation gaps in core and non-core teams, while also comparing gender diversity across projects.

1 Introduction

Open-source software (OSS) plays a major role in how modern software is developed. Platforms such as GitHub provide a way for developers around the world to access, modify and share projects. OSS is typically promoted as being community-focused where open collaboration, inclusiveness and transparency is prioritized. However, in practice, participation is not always equally distributed, especially when it comes to gender representation.

Several studied have shown that women and gender minorities are still significantly underrepresented in OSS. Moreover, these gaps are also reflected in the types of roles developers hold within the community. In most projects, there is a distinction between core and non-core contributors. Core contributors are generally responsible for a large portion of the codebase and often play an key role in making important decisions that guide the direction of the project. In contrast, non-core contributors usually make fewer contributions that tend to have less influence on the overall development of the project. New contributors typically begin in non-core roles and can transition into core roles over time as they become more active and involved in the project. As a result, the status of core contributor is usually held by those who joined the project early. Since software development has been historically a male-dominated field, the people who were around at the start of many projects were mostly men. While participation from women and gender minorities in OSS has grown, this imbalance is still noticeable today. Gaining influence and recognition can be significantly more difficult for women and gender minorities, even when they are contributing regularly and doing high-quality work.

To better understand this issue, we developed a plugin that analyzes GitHub repositories to detect gender representation gaps, specifically comparing core and non-core contributor roles. It is implemented as a Chrome extension designed to be accessible to anyone interested in understanding current dynamics within any GitHub project. The plugin infers gender using multiple sources of information, including *genderComputer* [15], a widely used tool that determines gender based on first names and geographic locations. Contributors are then classified into core and non-core groups based on their level of involvement within the project, more specifically their frequency of commits relative to other contributors. Moreover, to better understand the relationship between gender and contributor role, we conducted a series of statistical analyses. We used the *Blau Index* to analyze gender diversity within teams and applied *Fisher's Exact Test* to measure the association between gender and contributor role. Our aim with this tool is to spark discussion and bring more attention to gender representation issues in OSS communities.

The rest of the paper is organized as follows: Section 2 describes the design and functionality of the plugin, including the data extraction process; Section 3 explains the approach used for gender inference; Section 4 reviews how contributor roles are classified in related research and justifies the method used in this study; Section 5 presents the metrics considered for data analysis; and Section 6 provides conclusions and suggests possible directions for future work.

2 Plugin Design and Functionality

The plugin is designed to be an intuitive and user-friendly tool that allows anyone, regardless of technical background, to analyze gender representation in core and non-core teams in a given GitHub repository. The inference of gender and team composition is based on the most recent N commits, where N is a number given by the user though the plugin interface.

Using a commit-based approach provides a more consistent and predictable way to control the amount of data analyzed, compared to a time-based approach where the given time range might result in either too little or too much data. For instance, some repositories might have minimal activity in a given time-frame, resulting in insufficient data, while others might have an overwhelming amount of commits, leading to increased processing time. By using a fixed number of commits instead, the plugin ensures that the workload remains manageable and the results are comparable across repositories of different sizes and activity levels.

2.1 User Interface

The front-end of the plugin is implemented as a Chrome extension, providing a user-friendly interface for analyzing contributions in GitHub repositories. It is built using standard web technologies: HTML for defining the layout, CSS for styling and JavaScript for adding functionality. The extension is configured through a *manifest.json* file that defines key properties such as its name, version, permissions and behavior. When the user clicks the extension icon, it opens a popup window (*popup.html*) which acts as the main interface for user interaction.

The interface allows the user to enter a GitHub repository (as a *"owner/repo"* string format) and specify how many recent commits should be analyzed. If the user interacts with the extension while viewing a GitHub repository page, the repository field is automatically pre-filled with the corresponding *owner/repo* value by extracting it from the current tab's URL. When the user clicks the "Analyze" button, the extension sends a *GET* request to the backend containing the repository name and the desired commit count as query parameters. A loading animation is displayed during the data extraction process, after which the extension dynamically updates the UI with the information received from the backend.

2.2 Data Extraction

The Chrome extension communicates with a *FastAPI* backend, which is responsible for processing user input, retrieving data from GitHub and returning information back to the extension. More specifically, the backend receives as input a GitHub repository (a string in the format *"owner/repo"*) and the maximum number of commits to analyze, and outputs:

- the number of commits actually analyzed (since some repositories may have fewer commits than requested),
- the date of the oldest commit considered in the analysis,
- the gender distribution of contributors as percentages (female, male, non-binary or unknown),
- the number of core contributors by gender,
- the number of non-core contributors by gender,
- the Blau Index for the core team and non-core team,
- the average Blau Index for core and non-core teams across all previously analyzed repositories,
- the total number of repositories analyzed prior to the current one.

Notably, the core and non-core contributor groups are reported using counts rather than percentages since core groups are sometimes very small and percentages might not be as informative.

When the backend receives a request, it begins by extracting commit data using the *GitHub REST API* through the endpoint: *https://api.github.com/repos/owner/repo/commits*. A paginated commit history is returned, with each page containing up to 100 commits instead of the default 30 in order to speed up the API calls.

All relevant contributor data is stored using a *defaultdict* defined as:

```
data = defaultdict(lambda: {
    "pronouns": None,
    "bio": None,
    "name": None,
    "email": None,
    "location": None,
    "commits": 0
})
```

For each commit the script extracts the commit author's login/username and email (if it is not a *noreply.github.com* address which indicates that the email is private) and increments the commit count for the corresponding contributor. Additionally, the commit date is stored in order to keep track of the most recent commit included in the analysis.

After collecting all the usernames from the commit data, the backend retrieves additional profile information through the *GitHub GraphQL API* by using the following GraphQL query:

```
    query($login: String!) {
    user(login: $login) {
        pronouns
        bio
        name
        location
    }
}
```

The reason GraphQL is used instead of relying only on the GitHub REST API is because the REST API does not provide certain user information, such as *pronouns*, which is required later for the gender inference step. To improve efficiency, all GraphQL requests are made concurrently using Python's *asyncio* along with *httpx.AsyncClient*. Instead of querying users one at a time, multiple asynchronous tasks are launched in parallel, which significantly reduces the overall processing time.

3 Gender Inference

3.1 Existing Tools

There are several widely used tools for inferring gender, including *Genderize.io*, *Gender-API* and *NamSor*. These tools typically rely on first names and sometimes additional information, such as geographic location or language, to predict the most likely gender of an individual.

One tool that has gained significant popularity in the software development research community is *genderComputer* [1], developed by Vasilescu et al. [15]. It was originally created for a study on gender representation on Stack Overflow and has since been used in several other studies focused on open-source software (OSS) projects [3, 11, 13, 4]. The tool determines an individual's gender based on their first name and country information if provided, returning either *"female"*, *"male"* or *None* if the gender cannot be inferred. It processes a *(name, country)* tuple by first normalizing the name (e.g., converting "w35l3y" to "Wesley") and then checking against country-specific name lists (e.g. recognizing Andrea as a male name in Italy but a female name in Germany). In one study [16] that applied *genderComputer*, researchers were able to infer the gender of 873,392 GitHub contributors, which accounted for about 32.6% of the total group. Although this percentage might appear to be low, it corresponds to 80% of users who shared their real names. These results confirm that *genderComputer*'s accuracy is comparable to other popular gender inference tools, making it a reliable choice for similar studies.

[1]https://github.com/tue-mdse/genderComputer

3.2 Implementation

The plugin is designed to infer gender by using multiple sources of information to improve accuracy and respect how users choose to present themselves. GitHub allows users to add pronouns directly to their profiles, therefore the first and most important step is to check if they are given. Since pronouns are an explicit way for users to communicate their gender identity, the plugin gives priority to this information whenever it is available. A simple pattern-matching approach is used by checking for common pronouns such as *"she"*, *"her"* and *"hers"* which typically indicate a female identity, *"he"*, *"him"* and *"his"* which suggest a male identity, and gender-neutral pronouns like *"they"*, *"them"* and *"theirs"*. If no pronouns are found in this field, the plugin then analyzes the user's biography section. Although it is less common, some users choose to include personal information such as pronouns in their bio text. In those cases, the same pattern-matching logic is applied to identify any potential pronouns.

By focusing on the pronouns and biography first, the plugin reduces the need to rely on inference methods that could be less accurate or might misrepresent a user's gender identity. When neither of these fields provides enough information, the plugin falls back on *genderComputer*. As mentioned in section 3.1, *genderComputer* attempts to infer gender based on the user's first name, and when available, also uses their location to improve accuracy. If no name is available, the plugin makes one final attempt by analyzing the prefix of the user's email address.

In the end, the plugin groups users into four categories based on the inferred gender: *female*, *male*, *non-binary* and *unknown* (users for which none of the available information could reliably determine their gender identity).

4 Team Composition Inference

4.1 Contributor Roles

In open-source software (OSS) development, contributors are often classified according to their level of involvement within the project.

The *"onion model"* introduced by Nakakoji et al. [10], visualizes the OSS community as a set of nested layers, similar to the structure of an onion. Each layer corresponds to a different type of contributor based on their influence within the community, with roles closer to the center holding greater influence. The model defines eight roles, ranging from passive users and testers to active developers who regularly contribute code. However, recent research often reduces contributor classification to just two primary groups. Although the terminology varies across studies, with labels such as *"core" vs "peripheral"* [14, 17] and *"core" vs "non-core"* [2, 8, 1], the underlying definitions are similar in meaning.

Core contributors are typically the most active within a project, making consistent contributions over a long period of time. They often dedicate significant time and effort into submitting complex code changes, such as implementing new features or improving existing ones. These developers usually have a deep understanding of the project's architecture and may also have commit access, allowing them to make direct changes to the main codebase [9]. In contrast, *peripheral* or *non-core* contributors usually participate less frequently, making occasional contributions such as small patches or bug fixes. They tend to have a more limited understanding of the project and are generally not involved in important decision-making. Even though their level of involvement is lower, they make up most of the community and can transition into core roles over time.

For the purpose of this analysis, the *core/non-core* distinction is used due to its simplicity and intuitive meaning, making it more accessible for those not familiar with existing research terminology.

4.2 Classification Methods

Various approaches have been explored in literature to identify core contributors, including analyzing metrics such as the number of commits, the lines of code (LOC) contributed and contributor access levels, among others [18, 7].

The commit-based method defines core contributors as those who author the most commits in a repository. It follows the Pareto principle, also known as the 80-20 rule, which suggests that 80% of consequences result from 20% of causes [5]. In the context of OSS projects, this typically means that a small group of contributors is responsible for most of the work. For instance, Mockus et al. [9] found that around 20% of developers, referred to as the core group, produce approximately 80% of the total commits. However, the method does not take into account the size or complexity of each commit, since fixing a small typo is counted the same as adding a new feature. Despite this limitation, it remains one of the most widely used approaches due to its simplicity and clear results.

The LOC-based method classifies contributors by the amount of code changes made, such as lines added, lines deleted or a combination of both. Although it provides a way to capture the size of contributions instead of just their frequency, this method can still be biased. For instance, large changes from code refactoring or generated files might overshadow smaller but more meaningful contributions like critical bug fixes.

The access-based method identifies core developers based on whether they have write access to the repository. While this approach can reveal certain responsibilities assigned in the project, it does not necessarily indicate a contributor's level of involvement. For example, a contributor who regularly submits pull requests might not be considered core if they do not have write access. Meanwhile, someone with write access who mainly merges changes or makes minor edits could be classified as core, despite contributing less overall.

4.3 Implementation

This study follows a commit-based approach to distinguish between core and non-core contributors, given that it is widely adopted in existing research and allows easier comparison with previous work. Important to note is that users in the *unknown* category are still included to ensure that the classification reflects actual contribution levels. For example, if we were to exclude the commits of those users, we could end up with a woman being classified as part of the core team, even though someone with unknown gender actually contributed more.

Contributors are first sorted in descending order based on the number of commits they authored. The cumulative commit count is then calculated, and contributors are included in the core team until their combined commits account for at least 80% of the total. To ensure fairness, any contributors with the same number of commits as the last person included are also classified as core. The remaining contributors that fall below the threshold are classified as non-core.

However, this approach can sometimes result in a very large core team when many contributors have the same number of commits as the last person added to the core team. To mitigate this to some extent, we introduce an additional check for contributors who are tied at the cutoff. After identifying the core team based on the 80% threshold, we evaluate the impact of excluding all tied contributors. If removing them still keeps the core team's total commits above 70%, then we choose the option (including or excluding tied contributors) that results in a cumulative commit closer to the 80% target. This approach helps keep the core team size more reasonable without significantly affecting the total contribution percentage.

5 Data Analysis

For data analysis, we consider only the *female*, *male* and *non-binary* categories in order to more accurately reflect gender representation and to prevent the *unknown* category from skewing the results.

5.1 Blau Index

The *Blau Index* [12] is a commonly used measure of diversity that determines the probability that two individuals selected at random from a population will belong to different categories. The index is calculated as:

$$BI = 1 - \sum_{i=1}^{n} p_i^2$$

Where:

- p_i is the proportion of the population in the i-th category

- n is the total number of categories

The maximum Blau index depends on the number of categories n and is computed as $1-\frac{1}{n}$. To make values comparable across groups with different numbers of categories, the index can be normalized by dividing it by the maximum value:

$$BI_{normalized} = \frac{BI}{BI_{max}} = \frac{BI}{1-\frac{1}{n}}$$

This normalization scales the Blau index to a value between 0 and 1, where 0 means that all individuals are in the same category (no diversity) and 1 means that all individuals are in different categories (maximum diversity).

Although this study involves only three categories (female, male, non-binary), normalization is applied to present the diversity scores on a standardized scale. The script computes two scores, one for the core team and one for the non-core team, which are saved to a *.csv* file along with the *datetime* of the analysis, the repository name and the number of commits analyzed. If the file contains previous entries, the script sorts them by date and keep only the latest entry for each repository, excluding also the current repository to avoid self-comparison. Based on the remaining data, it calculates the average diversity scores for core and non-core teams. These averages allow users to compare the current repository's diversity scores with those of other repositories where the plugin has been used.

5.2 Fisher's Exact Test

Fisher's Exact Test [6] is used to determine if there is a significant association between two categorical variables. It is often used when the sample sizes are small, which could impact the results of other tests such as the Chi-Square Test. The analysis is based on a contingency table of observed frequencies for each combination of categories, as seen in Table 1.

	Category 1	***Category 2***
Group 1	a	b
Group 2	c	d

Table 1: Contingency table

In Python, the *scipy.stats.fisher_exact* function returns two values: the *odds ratio* and the *p-value*.
The *odds ratio* is calculated as:

$$OR = \frac{a \times d}{b \times c}$$

An odds ratio of 1 suggests that there is no association between the two categorical variables, while values greater than 1 indicate a positive association (the odds of being in Category 1 are higher in Group 1 than in Group 2) and values less than 1 indicate a negative association (the odds of being in Category 1 are lower in Group 1 than in Group 2).

The *p-value* is computed as:s

$$P = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!b!c!d!(a+b+c+d)!}$$

A small p-value (typically less than 0.05) rejects the null hypothesis, suggesting that the association is statistically significant.

In this study, Fisher's Exact Test is used to determine if there is an association between gender and team role (core or non-core). To meet the 2×2 table requirement, the female and non-binary categories are combined into a single group, since non-binary counts are usually very small. Similarly to the Blau Index, the odds ratio and the p-value for each repository are also stored in the *.csv* file.

6 Conclusion and Future Work

In conclusion, this study highlights the gender gaps that exist in open-source software (OSS) projects, specifically when it comes to contributor roles. The developed plugin offers an accessible way to visualize and compare gender representation across repositories, helping to raise awareness to this ongoing issue. While the tool has some limitations, it provides a practical foundation for further research on diversity within OSS communities.

One limitation of the current commit-based approach is that it can lead to a relatively large core team when multiple contributors have the same number of commits as the last person included. For future work, we plan to address this by incorporating lines of code (LOC) as a secondary sorting criterion. Contributors would first be sorted by their total number of commits and ties would then be handled using the LOC metric (calculated as net LOC = Lines Added - Lines Deleted). This method was not implemented initially because collecting LOC data requires an additional API call for every single commit, which would significantly slow down the plugin's processing time.

Regarding statistical tests, the user interface currently displays the Blau Index for both core and non-core teams along with average scores computed from other repositories. As a next step, we aim to also integrate the results of the Fisher's Exact Test such that the odds ratio and p-value are presented in a user-friendly manner. Additionally, we want to explore how to average the Fisher's Exact Test results from different repositories. Since odds ratios are not additive, one option would be to apply a logarithmic transformation before averaging. For p-values, Fisher's combined probability test could be used to analyze the overall significance of multiple test results.

The processing speed of the plugin is heavily influenced by the number of contributors that need to be analyzed. Even when extracting the same number of commits from different repositories, the time it takes to process them can vary significantly depending on how many contributors are involved. This is because the plugin performs gender inference on each individual contributor, which is time-consuming even when tasks are executed in parallel. For future work, it would be worth exploring the use of a high-performance computing (HPC) cluster to further reduce processing time for repositories with very large contributor groups.

References

[1] Caius Brindescu, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. An empirical investigation into merge conflicts and their effect on software quality. *Empirical Software Engineering*, 25:562–590, 2020.

[2] Simon Butler, Jonas Gamalielsson, Björn Lundell, Per Jonsson, Johan Sjöberg, Anders Mattsson, Niklas Rickö, Tomas Gustavsson, Jonas Feist, Stefan Landemoo, et al. An investigation of work practices used by companies making contributions to established oss projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 201–210, 2018.

[3] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. Gender diversity and women in software teams: How do they affect community smells? In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 11–20. IEEE, 2019.

[4] Shamse Tasnim Cynthia and Banani Roy. An empirical study on the impact of gender diversity on code quality in ai systems. *arXiv preprint arXiv:2505.03082*, 2025.

[5] Rosie Dunford, Quanrong Su, Ekraj Tamang, and Abigail Wintour. The pareto principle. *The Plymouth Student Scientist*, 7(1):140–148, 2014.

[6] Jenny V Freeman and Michael J Campbell. The analysis of categorical data: Fisher's exact test. *Scope*, 16(2):11–12, 2007.

[7] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 164–174. IEEE, 2017.

[8] Umme Ayda Mannan, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. On the relationship between design discussions and design quality: a case study of apache projects. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 543–555, 2020.

[9] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[10] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*, pages 76–85, 2002.

[11] Marco Ortu, Giuseppe Destefanis, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. How diverse is your team? investigating gender and nationality diversity in github teams. *Journal of Software Engineering Research and Development*, 5:1–18, 2017.

[12] Antonio Solanas, Rejina M Selvam, José Navarro, and David Leiva. Some common indices of group diversity: Upper boundaries. *Psychological Reports*, 111(3):777–796, 2012.

[13] Sayma Sultana, Asif Kamal Turzo, and Amiangshu Bosu. Code reviews in open source projects: how do gender biases affect participation and outcomes? *Empirical Software Engineering*, 28(4):92, 2023.

[14] Antonio Terceiro, Luiz Romario Rios, and Christina Chavez. An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In *2010 Brazilian Symposium on Software Engineering*, pages 21–29. IEEE, 2010.

[15] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. Gender, representation and online participation: A quantitative study of stackoverflow. In *2012 International Conference on Social Informatics*, pages 332–338. IEEE, 2012.

[16] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 3789–3798, 2015.

[17] Wenxin Xiao, Hao He, Weiwei Xu, Yuxia Zhang, and Minghui Zhou. How early participation determines long-term sustained activity in github projects? In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 29–41, 2023.

[18] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E Hassan, and Naoyasu Ubayashi. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th international workshop on principles of software evolution*, pages 46–55, 2015.