Contribution-Based Firing of Developers?

Vincenzo Orrei
REVEAL @ Software Institute – USI, Lugano, Switzerland
vincenzo.orrei@usi.ch

Marco Raglianti
REVEAL @ Software Institute – USI, Lugano, Switzerland
marco.raglianti@usi.ch

Csaba Nagy
REVEAL @ Software Institute – USI, Lugano, Switzerland
csaba.nagy@usi.ch

Michele Lanza
REVEAL @ Software Institute – USI, Lugano, Switzerland
michele.lanza@usi.ch

ABSTRACT

There has been some recent clamor about the developer layoff and turnover policies enacted by high-profile corporate executives. Precisely defining the contributions in software development has always been a thorny issue, as it is difficult to establish a developer's "performance" without recurring to guesswork, due to how software development works and how Git persists history.

Taking inspiration from a seemingly informal notion, the *pony factor*, we present an approach to identify the key developers in a software project. We present an analysis of 1,011 GitHub repositories, providing fact-based reflections on development contributions.

CCS CONCEPTS

• **Software and its engineering** → Collaboration in software development; • **Human-centered computing** → *Collaborative and social computing*.

KEYWORDS

Contribution, pony factor, developer performance, mining

ACM Reference Format:
Vincenzo Orrei, Marco Raglianti, Csaba Nagy, and Michele Lanza. 2023. Contribution-Based Firing of Developers?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3611643.3613085

1 INTRODUCTION

Since the takeover of Twitter by Elon Musk, the company has seen a considerable reduction in its developer workforce. Nonetheless, the system seems resilient to such a drain, which raises the question, "who are the key developers?", leading to others: Who are the unlucky disposable ones? Are they disposable because they did not contribute enough? What is a developer's contribution anyway? How does removing a bug compare to adding proper documentation? All such questions boil down to how someone, billionaire or not, can confidently pronounce, "You're fired!".

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3613085

Productivity, contribution, and code ownership are related concepts relevant both in an industrial setting and for open source communities [26]. They tie into a system's dependence on certain developers [6, 8, 20, 24]. This led to the question, "If Guido was hit by a bus?",[1] posed in 1994 in the Python mailing list, due to the project's dependency on Guido van Rossum: What destiny would the Python project have if he simply "walked away"?

While we recognize the importance and relatedness of code ownership and productivity, we focus solely on the concept of contribution: We present a large-scale study performed on 1,011 GitHub projects, leveraging the seemingly simple metric called the "Pony Factor"[2] (*PF*) to identify the key contributors in software projects. *PF* enumerates the smallest possible set of developers, called *ponies*, who contributed more than 50% of the total commits.

The Pony Factor. First defined by Gruno and taken up by Nalley,[3] the *"PF shows the diversity of a project in terms of the division of labor among committers […]. The higher the PF, the more resilient the project is towards contributors leaving or taking a vacation".*

The PF is given by:

$$PF = min(n | n \in \{1, \ldots, N\} \text{ and } \sum_{i=1}^{n} C_i \geq K \times V)$$

where *PF* is the pony factor, N is the number of developers, C_i is the number of commits for the i-th developer (with developers sorted in decreasing order of number of commits), K is the percentage to cover (typically $K = 50\%$), and V is the total number of commits.

Our investigation led to several reflections and insights. We will show how small modifications in the metrics can significantly affect the key developers identified by the pony factor. The objective is to show that, even if refined, it is unreliable to define a developer's performance with simple metrics, if not properly contextualized.

2 RELATED WORK

Defining a reliable contribution metric is not straightforward [18, 23], and there is no consensus on how it should be done [19].

Trucks, Buses, and Heroes. Ricca and Marchetto studied 20 randomly selected open-source projects [28], computing the truck factor, *i.e.,* "the number of people on your team that have to be hit with a truck/bus before the project is in serious trouble" [36]. They found that hero developers (*i.e.,* developers that exclusively manage/own a number of files $\geq \alpha\%$ of the total) were common in the considered projects and that the truck factor was generally low. Avelino *et al.* proposed a heuristic-based approach to estimate a system's truck factor [3], confirming a low truck factor for 65% of 133 popular projects on GitHub.

[1] See **https://tinyurl.com/if-guido** [acc. September 23, 2023]
[2] See **https://tinyurl.com/px32w868** [acc. September 23, 2023]
[3] See **https://tinyurl.com/24b9n6bw** [acc. September 23, 2023]

Cosentino *et al.* computed the bus factor on files, directories, and branches for Git-based development [7] to perform a risk assessment. Jabrayilzade *et al.* [17] proposed a multimodal bus factor algorithm, integrating history in the version control system, code reviews, and meetings meta-data.

The truck factor itself has seen several slightly different definitions [10, 11, 15, 29]. It has been used in various contexts, such as identifying experts [2, 35] and quantifying knowledge loss that would be incurred if they left the project [16, 30]. Agrawal *et al.* [1] identified "hero projects" in which less than 20% of developers produce more than 80% of the code. They analyzed the impact of hero developers on a project as an alternative to the bus factor.

Code Ownership. Studies on code ownership practices have often led to contrasting results. For example, Kurapati *et al.* surveyed 109 practitioners through LinkedIn, Yahoo, and Google Groups. 50% of respondents used collective code ownership [22]. Similarly, Rauf *et al.* found that 43% of developers from 45 Pakistani software companies used collective code ownership [27]. On the contrary, Rodriguez *et al.* found that collective code ownership was the least-used agile practice among 408 surveyed Finnish software practitioners from 200 organizations [31]. In large projects, files are often owned by multiple developers. Sindhgatta *et al.* studied the evolution of an enterprise system developed following agile practices. They found that 60% of the files had more than one owner [32].

Attributing ownership is relevant not only for project resilience but also for software quality. Bird *et al.* examined the relationship between ownership and software quality in Windows Vista and Windows 7 [5]. They defined the proportion of ownership of a contributor for a component as the ratio of the number of commits by a contributor relative to the total number of commits for that component. They considered major contributors who had at least 5% ownership. Greiler *et al.* replicated Bird *et al.*'s study on four Microsoft systems at a smaller granularity level: Files instead of components [14], confirming that code ownership correlates with code quality. Foucault *et al.* explored the relationship between ownership metrics and fault-proneness in open-source projects [12]. They did not find a correlation between ownership and module fault metrics, due to the distribution of contributions among developers and the presence of "heroes". For most projects, they found a single "hero" performing most of the commits.

Most works (except [1]) focus on a limited number of systems (*e.g.,* [7, 17, 28]) and various evolving definitions of the bus factor. Our main contribution is a large-scale study on GitHub repositories leveraging the pony factor to derive data-driven insights.

3 DATASET CREATION

Figure 1 provides an overview of our dataset creation process. We used *SEART-GHS* [9], with the filtering criteria shown in Figure 1, to create an initial dataset of 2,053 GitHub repositories, with almost a million developers who produced a billion lines of code.

3.1 Data Cleaning

The extraction process is hampered by data quality problems, which calls for the following data cleaning steps.

Small project removal. We excluded projects with less than 10K lines of code in their latest revision.

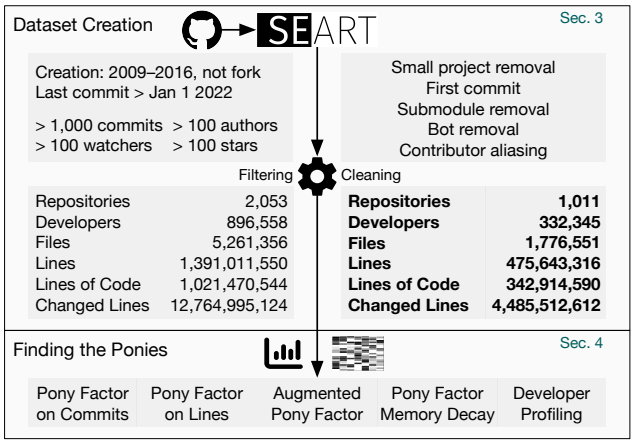

Figure 1: Dataset Creation and Analysis Process.

First commit. Not all projects started their development on GitHub. Some projects can have a large initial commit hiding earlier development history and distorting the pony factor calculation. Thus, we filter projects whose initial history is not in Git, by excluding repositories with a first commit over 10K lines of code in source files (*i.e.,* excluding non-source files such as READMEs).

Submodule removal. Git supports submodules (*i.e.,* a repository as a subdirectory of another repository). Discriminating between a project split into submodules and a third-party library added as a submodule is out of the scope of the present work. Therefore, we exclude projects with submodules in their development history.

Bot removal. Bots are popular in GitHub repositories [13, 33] and can generate noise in our analysis. We implemented an approach to detect bots by searching for recurring patterns in the contributors' usernames (*e.g.,* 'GitHub Action', '*name*[bot]'), including a recent bot list identified by Golzadeh *et al.* [13], and then remove their commits from the project.

Contributor aliasing. Who are the developers? Git tracks commit authors by their names and email addresses, which are then matched with users on GitHub. The same person can use multiple names, email addresses, or even GitHub accounts.

This calls for the need to *aliasing* the contributors. Various approaches exist in the literature [4, 25, 34]. We implement a strategy with: *(i)* contributors' names Levenshtein similarity greater than 0.9 *or (ii)* emails' usernames Jaro-Winkler similarity greater than 0.9 with a prefix weight of 0.1 *and* name Levenshtein similarity greater than 0.7. Figure 2 shows contributors before and after aliasing.

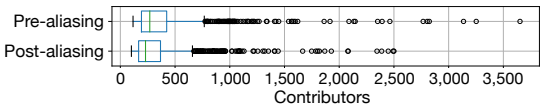

Figure 2: Contributors Pre- and Post-Bot-Alias Removal.

A significant amount of aliases would have negatively impacted the relevance of identified ponies. The combined effect of bot exclusion and aliasing leads to a considerable reduction in contributors in the range of 2.2%–44.9% for the single repository (mean: 13.2%, std: 6.8%). For example, the *gcc-mirror/gcc* repository dropped from 3,652 to 2,082 contributors, with a 43.0% reduction.

3.2 Final Dataset

The final dataset consists of 1,011 repositories with 332K contributors, who changed a total of 4.5 billion lines, leading to a total count of 343 million lines of code in the repositories at mining time. The dataset is available in the replication package.[4]

4 FINDING THE PONIES

Figure 3 shows how PF is distributed across our dataset. To make the figure more clear, we do not display observations beyond the 99-*th* percentile in at least one of the four distributions (28 outliers).

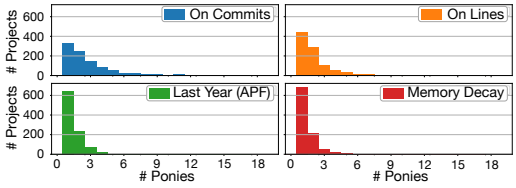

Figure 3: Ponies On Commits, On Lines, On Lines Last Year (Augmented Pony Factor), and With Memory Decay.

The top-left side shows the PF calculated following the original definition based on the number of commits. We calculated PF based on the changed lines within the commits in the top-right subplot. In the bottom-left, we narrowed it down to the commits only in the last year (before the last known commit), and in the bottom-right, we calculated the PF according to the memory decay on the lines.

4.1 Pony Factor On Commits

As the top-left chart in Figure 3 shows, most projects have a low number of ponies considering the original commit-based definition: 99.3% have less than 25 ponies (med: 2, mean: 3.42, std: 5.22). Projects in the dataset had contributors ranging from 101 to 2,500 (med: 230, mean: 328.73, std: 301.00). Looking at the normalized numbers (*i.e.,* the ponies over the total number of contributors), on average, 1.24% of the contributors are ponies (std: 1.22).

4.2 Pony Factor On Lines

An important limitation of measuring the pony factor based on commits is that it does not consider the magnitude of the contribution in the commits. We address this limitation by analyzing the changed lines in each commit and refining the granularity of PF to line changes so that in the PF equation V becomes the total number of changed lines and C_i the number of changed lines for the i-th developer. Since lines cannot be interpreted in binary files, we narrow the scope to non-binary files.

Figure 3 (top-right) shows how the distribution of ponies gets progressively steeper towards lower values (*i.e.,* fewer ponies) when counting contributed lines. 99.5% of the projects have less than 12 ponies. The average ratio of ponies drops significantly to 2.23% of the contributors (std: 1.86). The *prebid/prebid.js* project has 450K lines of code, including several small adapters added by separate developers, resulting in many ponies (82) at commit level. We identified 10 ponies at line granularity. They are the key developers who oversee the larger codebase.

[4] https://doi.org/10.6084/m9.figshare.23989335

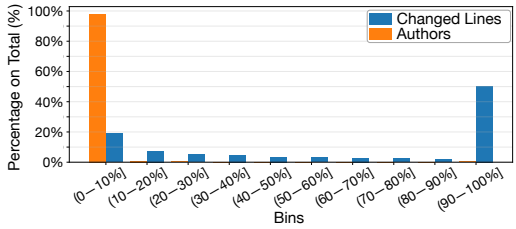

Figure 4: Contribution Distribution.

Figure 4 shows a bar chart of contributors based on their changed lines of code, normalized according to the top developer's contribution, and distributed in ten bins. For example, the first bin includes developers who committed between 0–10% of the top developer's contribution, and so on. The numbers are the mean values among all projects. Most of the contributors (~98%) are in the first bin for an average project. Their contributions amount to less than a fifth of the total changed lines. The top developers' bin is two and a half times higher but achieved only by 1% of the authors. Very few developers are often responsible for the vast majority of the contribution, corroborating the concept of hero developers.

4.3 Augmented Pony Factor

Gruno and Nalley modified the PF formula to take into account only *active* developers: The *Augmented Pony Factor* (APF). As there is no operational definition to discern "active" developers, we assess this formula with active developers who had at least one commit in the last year (preceding the last commit in the analyzed project).

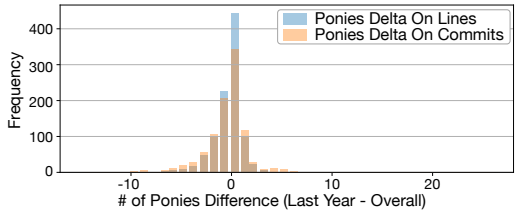

Figure 5: Augmented Pony Factor.

Figure 5 shows the deltas of the APF formula on the PF after outliers exclusion (as done for Figure 3). For the lines, the number of ponies drops with a mean of -0.63 (med: 0.0, std: 1.61), while for commits, the effects show more spread with a minimum of -71 and a maximum of 26 new ponies (mean: -0.90, med: 0.0, std: 4.08). Narrowing the dataset to the last year of activity shows a median reduction of the total contribution to 4.69% (mean: 9.53, std: 1.28) for code lines. This implies that the calculation of the APF takes into account less than 5% of the total contributions. The recent activity has presumably decreased for these projects.

4.4 Memory Decay

Another limitation of the original PF definition is that it counts all contributors, regardless of whether they are still active in the project. Following the approach of Krüger *et al.* [21], we leverage the Ebbinghaus forgetting formula: $R = e^{-\frac{t}{s}}$, where R is the retention rate, the recall rate of a given event, t is the time elapsed in days since the event, and s is the *memory strength.*

We implemented a pony factor version that considers only the changed lines that developers did not yet forget. So, the recalled contribution for the single developer is: $C_{md} = \sum_{i=1}^{n} LOC_i \times R(t_i)$. Where n is the total number of contributions, LOC_i is the number of the lines of code of the *i-th* contribution, and R is the retention rate, which depends on the time of contribution t_i.

Figure 3 (bottom-right) shows how ponies are distributed with this restriction. The number of ponies drops for most projects: 99.3% have less than 7 ponies. There are also projects that increase their PF with memory decay. In Figure 6, we compare PF with and without memory decay and sort projects according to their change in the number of ponies. 119 (11.8%) projects have more ponies, showing potentially increased activity and distribution of key developers. 443 (43.8%) remain stable with the same number of ponies. The number of ponies decreases for 449 (44.4%) projects. Memory decay weighs more recent history than overall contribution, significantly changing who is identified as a relevant contributor.

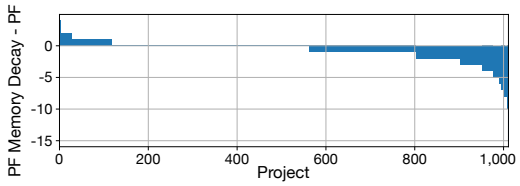

Figure 6: Pony Factor With and Without Memory Decay.

The PF drops to 3 in the *prebid/prebid.js* project showing two key developers who started contributing intensively. The ponies changed 34K lines (20K, 12K, and 2K, respectively) within the memory decay time window. Focusing on recency is risky in long-lived large projects: Is someone a pony if they contributed 1,000 lines to a million-line project within the last year? Hardly so. Also, there is no clear way of establishing that someone is still active. The heuristic "performed at least a commit within the last year" is rough and intrinsically unreliable.

4.5 Developer Profiling

Another factor that must be considered is that despite obtaining a finer-grained, line-based definition of the pony factor, there is no distinction between the type of contribution developers make. For example, a line of code in Java is treated the same as a line of code in C++ or Markdown. We developed a custom visualization, named contribution heat map, depicted in Figure 7.

The figure shows the contribution of the *apache/drill* project. The upper part shows a contributor for each line, and for each column, the file types in the repository are sorted from most to least changed. In the specific project, the most changed files are .java files, followed by grouping files without extension and .json files. Each tile in the heat map is colored (from light blue to dark blue) according to the total contributions of a person concerning the file type. The contributors are sorted according to their line-based contributions. Because the predominance of certain file types obscures less "popular" file types, the bottom half of the figure shows the heat map again, but with the coloring relative to each file type. The horizontal red line divides ponies from non-ponies.

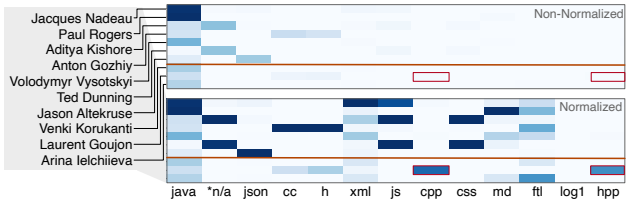

Figure 7: Contribution Heat Map of the *apache/drill* Project.

We see, for example, that while the top contributor is a .java pony, there is someone (Laurent Goujon) who globally is not detected as a pony, but when it comes to .cpp and .hpp, he is undoubtedly a key developer. The contributor heat map helps to inspect what would be obscured if one only looks at the big picture.

4.6 Data quality

The analyses conducted are strongly influenced by the quality of the data stemming from Git. Some projects such as *kodecocodes/swift-algorithm-club* are not real systems and it is impossible to identify them systematically. Still others, such as *elastic/ansible-elasticsearch*, although they have seen activity over the past year, this only amounts to 400 lines of code and explicitly states "*This project is no longer maintained*" in their README.

From Git's history, it is not possible to derive *pair programming activities* and, in any case, it would then be non-trivial to assign a consistent contribution value to each developer.

The Git algorithm to identify changed lines ignores the quality of the change made. For example, a single modification, such as re-formatting, will consist of as many modified lines as those in which the reformatting is substantiated, overestimating the contribution.

5 CONCLUSIONS

We presented an early-stage approach to identify potential key developers in projects, leveraging the "pony factor" metric. Although variations of the original definition based on commits lead to quite different results, these are promising. Our approach suffers from a number of known problems: Aliasing can only be tackled partially, and Git and GitHub generally suffer from data loss problems (*e.g.*, commit squashing) which are difficult to address. Moreover, many existing approaches do not distinguish between the types of contribution. As the contribution heat map has shown, some crucial developers provide their contributions only in specific locations of the system. If one does not make a distinction, there is a substantial risk of underestimating those contributors' importance. Bots and cross-cutting commits without coding semantics also exacerbate the data quality problem. The ultimate issue, and part of our future work, is that there is no ground truth. The ponies that we detect are only potentially actual key developers. There is no way of saying whether they are still around or relevant, and given the many data quality problems, any detection of the ponies comes with a huge grain of salt. Essentially, anyone, even random billionaires, might say "You're Fired!", but doing so confidently is rather pretentious.

Acknowledgments. This work is supported by the Swiss National Science Foundation (SNSF) through the project "INSTINCT" (SNF Project No. 190113).

REFERENCES

[1] Amritanshu Agrawal, Akond Rahman, Rahul Krishna, Alexander Sobran, and Tim Menzies. 2018. We Don't Need Another Hero? The Impact of "Heroes" on Software Development. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2018)*. ACM, 245–253. https://doi.org/10.1145/3183519.3183549

[2] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who Should Fix This Bug?. In *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*. ACM, 361–370. https://doi.org/10.1145/1134285.1134336

[3] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A Novel Approach for Estimating Truck Factors. In *Proceedings of the 24th IEEE International Conference on Program Comprehension (ICPC 2016)*. IEEE, 1–10. https://doi.org/10.1109/ICPC.2016.7503718

[4] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining Email Social Networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR 2006)*. ACM, 137–143. https://doi.org/10.1145/1137983.1138016

[5] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't Touch My Code! Examining the Effects of Ownership on Software Quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, 4–14. https://doi.org/10.1145/2025113.2025119

[6] H. Alperen Cetin. 2019. Identifying the Most Valuable Developers Using Artifact Traceability Graphs. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. ACM, 1196–1198. https://doi.org/10.1145/3338906.3342487

[7] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Assessing the Bus Factor of Git Repositories. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2015)*. IEEE, 499–503. https://doi.org/10.1109/SANER.2015.7081864

[8] Charmayne Cullom and Richard Cullom. 2006. Software Development: Cowboy or Samurai. *Communications of the IIMA* 6, Article 1 (2006), 8 pages. Issue 2. https://doi.org/10.58729/1941-6687.1305

[9] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *Proceedings of the 18th IEEE/ACM International Conference on Mining Software Repositories (MSR 2021)*. IEEE, 560–564. https://doi.org/10.1109/MSR52588.2021.00074

[10] Mívian Ferreira, Guilherme Avelino, Marco Tulio Valente, and Kecia A. M. Ferreira. 2016. A Comparative Study of Algorithms for Estimating Truck Factor. In *Proceedings of the 10th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2016)*. IEEE, 91–100. https://doi.org/10.1109/SBCARS.2016.20

[11] Mívian Ferreira, Marco Tulio Valente, and Kecia Ferreira. 2017. A Comparison of Three Algorithms for Computing Truck Factors. In *Proceedings of the 25th IEEE/ACM International Conference on Program Comprehension (ICPC 2017)*. IEEE, 207–217. https://doi.org/10.1109/ICPC.2017.35

[12] Matthieu Foucault, Jean-Rémy Falleri, and Xavier Blanc. 2014. Code Ownership in Open-Source Software. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014)*. ACM, Article 39, 9 pages. https://doi.org/10.1145/2601248.2601283

[13] Mehdi Golzadeh, Damien Legay, Alexandre Decan, and Tom Mens. 2020. Bot or Not? Detecting Bots in GitHub Pull Request Activity Based on Comment Similarity. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW 2020)*. ACM, 31–35. https://doi.org/10.1145/3387940.3391503

[14] Michaela Greiler, Kim Herzig, and Jacek Czerwonka. 2015. Code Ownership and Software Quality: A Replication Study. In *Proceedings of the 12th IEEE/ACM Working Conference on Mining Software Repositories (MSR 2015)*. IEEE, 2–12. https://doi.org/10.1109/MSR.2015.8

[15] Christoph Hannebauer and Volker Gruhn. 2014. Algorithmic Complexity of the Truck Factor Calculation. In *Product-Focused Software Process Improvement (PROFES 2014)*. Springer, 119–133. https://doi.org/10.1007/978-3-319-13835-0_9

[16] Lile Hattori and Michele Lanza. 2009. Mining the History of Synchronous Changes to Refine Code Ownership. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories (MSR 2009)*. IEEE, 141–150. https://doi.org/10.1109/MSR.2009.5069492

[17] Elgun Jabrayilzade, Mikhail Evtikhiev, Eray Tüzün, and Vladimir Kovalenko. 2022. Bus Factor in Practice. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2022)*. ACM, 97–106. https://doi.org/10.1145/3510457.3513082

[18] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. In *Proceedings of the 39th International Conference on Software Engineering (ICSE 2017)*. IEEE Press, 164–174. https://doi.org/10.1109/ICSE.2017.23

[19] Gibeon Soares de Aquino Júnior and Silvio Romero de Lemos Meira. 2009. Towards Effective Productivity Measurement in Software Projects. In *Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA 2009)*. IEEE, 241–249. https://doi.org/10.1109/ICSEA.2009.44

[20] Sandeep Krishnamurthy. 2002. Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects. *First Monday* (2002). https://ssrn.com/abstract=667402

[21] Jacob Krüger, Jens Wiemann, Wolfram Fenske, Gunter Saake, and Thomas Leich. 2018. Do You Remember This Source Code?. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering (ICSE 2018)*. ACM, 764–775. https://doi.org/10.1145/3180155.3180215

[22] Narendra Kurapati, Venkata Sarath Chandra Manyam, and Kai Petersen. 2012. Agile Software Development Practice Adoption Survey. In *Agile Processes in Software Engineering and Extreme Programming (XP 2012)*. Springer, 16–30. https://doi.org/10.1007/978-3-642-30350-0_2

[23] Jalerson Lima, Christoph Treude, Fernando Figueira Filho, and Uirá Kulesza. 2015. Assessing Developer Contribution with Repository Mining-Based Metrics. In *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME 2015)*. IEEE, 536–540. https://doi.org/10.1109/ICSM.2015.7332509

[24] Steve McConnell. 2004. *Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers*. Addison-Wesley.

[25] David Moreno, Santiago Dueñas, Valerio Cosentino, Miguel Angel Fernandez, Ahmed Zerouali, Gregorio Robles, and Jesus M. Gonzalez-Barahona. 2019. SortingHat: Wizardry on Software Project Members. In *Companion Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE-Companion 2019)*. IEEE, 51–54. https://doi.org/10.1109/ICSE-Companion.2019.00036

[26] Ayushi Rastogi and Ashish Sureka. 2014. What Community Contribution Pattern Says about Stability of Software Project?. In *Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC 2014)*, Vol. 2. IEEE, 31–34. https://doi.org/10.1109/APSEC.2014.88

[27] Abdul Rauf and Mohammad AlGhafees. 2015. Gap Analysis between State of Practice and State of Art Practices in Agile Software Development. In *Proceedings of the 2015 Agile Conference*. IEEE, 102–106. https://doi.org/10.1109/Agile.2015.21

[28] Filippo Ricca and Alessandro Marchetto. 2010. Are Heroes Common in FLOSS Projects?. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2010)*. ACM, Article 55, 4 pages. https://doi.org/10.1145/1852786.1852856

[29] Filippo Ricca, Alessandro Marchetto, and Marco Torchiano. 2011. On the Difficulty of Computing the Truck Factor. In *Proceedings of the 12th International Conference on Product-Focused Software Process Improvement (PROFES 2011)*. Springer, 337–351. https://doi.org/10.1007/978-3-642-21843-9_26

[30] Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, and Audris Mockus. 2016. Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a Project at Avaya. In *Proceedings of the 38th International Conference on Software Engineering (ICSE 2016)*. ACM, 1006–1016. https://doi.org/10.1145/2884781.2884851

[31] Pilar Rodríguez, Jouni Markkula, Markku Oivo, and Kimmo Turula. 2012. Survey on Agile and Lean Usage in Finnish Software Industry. In *Proceedings of the 6th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2012)*. ACM, 139–148. https://doi.org/10.1145/2372251.2372275

[32] Renuka Sindhgatta, Nanjangud C. Narendra, and Bikram Sengupta. 2010. Software Evolution in Agile Development: A Case Study. In *Proceedings of the 25th ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA 2010)*. ACM, 105–114. https://doi.org/10.1145/1869542.1869560

[33] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW, Article 182 (2018), 19 pages. https://doi.org/10.1145/3274451

[34] Igor Scaliante Wiese, José Teodoro Da Silva, Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2016. Who is Who in the Mailing List? Comparing Six Disambiguation Heuristics to Identify Multiple Addresses of a Participant. In *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME 2016)*. IEEE, 345–355. https://doi.org/10.1109/ICSME.2016.13

[35] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi. 2015. 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 (IWPSE 2015)*. ACM, 46–55. https://doi.org/10.1145/2804360.2804366

[36] Nico Zazworka, Kai Stapel, Eric Knauss, Forrest Shull, Victor R. Basili, and Kurt Schneider. 2010. Are Developers Complying with the Process: An XP Study. In *Proceedings of the 4th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2010)*. ACM, Article 14, 10 pages. https://doi.org/10.1145/1852786.1852805

Received 2023-05-03; accepted 2023-07-19