Contribution-Based Firing of Developers?

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

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

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

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 unlucky disposable one? 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

The Pony Factor. First defined by Gruno and taken up by Nalley, 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:

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

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 ≥ α% of the total) were common in the considered projects and that the truck factor was generally low. Avellino 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.

Permiss to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ESEC/FSE ’23, December 3–9, 2023, San Francisco, CA, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9327-9/23/12...
$15.00
https://doi.org/10.1145/3611463.3613085

1See https://tinyurl.com/if-guido [acc. August 22, 2023]
2See https://tinyurl.com/px32w868 [acc. August 22, 2023]
3See https://tinyurl.com/24b9zn6bw [acc. August 22, 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, Kuwarapati 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. Sindigatta 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 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.

**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.

A significant amount of aliases would have negatively impacted the relevancy 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.

4.3 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 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.

4.4 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: \( R = e^{-\gamma t} \), 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.

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 kodicocodes/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-formating, 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).