Revisiting Code Authorship in Large FLOSS Projects

Arthur Pilone*, Marco Raglianti†, Michele Lanza†, Fabio Kon*, Paulo Meirelles*
*Institute of Mathematics, Statistics, and Computer Science — University of São Paulo, Brazil
†REVEAL @ Software Institute — USI, Lugano, Switzerland

***Abstract*—Authorship concentration metrics, such as the Truck Factor or Pony Factor, have been used to gauge a project's vulnerability to developer turnover. While it is well accepted that teams should strive to improve these metrics and avoid excessive knowledge concentration among a few developers, it remains unclear how the considered granularity and project characteristics (*e.g.,* size, language, age) affect the evolution of authorship concentration metrics at the file, module, and project levels: To what extent is it common for large, bazaar-like Free/Libre and Open Source Software (FLOSS) projects to have entire subcomponents with a high authorship concentration?**

We replicate Orrei *et al.*'s study "Contribution-Based Firing of Developers?" on the distribution of the Pony Factor authorship concentration metric using a new stratified sample of 80 large FLOSS projects. Additionally, we analyze the Pony Factor at the file level and subsequently aggregate it at the submodule and module levels, finding that most long-lived projects have over 80% of their submodules primarily authored by a single contributor. Besides confirming the prevalence of low Pony Factor values, our replication reveals how project-level authorship metrics overlook many of the nuances that characterize how knowledge concentrates and dilutes in complex real-world software projects, with obvious consequences for the long-term sustainability of large and relevant FLOSS systems.

***Index Terms*—Authorship Concentration, Code Ownership, Pony Factor, Software Maintenance, Open Source Software**

I. Introduction

Software engineering researchers have extensively explored the concept of code ownership over the past decades. Ample availability of authorship information mined from version control systems led them to investigate whether individually owned parts of a codebase tend to have a different incidence of faults [4], [11], [40], if code authorship can help to identify experts in specific system components [16], [22], [28], and if authorship metrics help to pinpoint knowledge concentration and subsystems at risk of developer turnover [15], [30], [35].

Both practitioners and researchers have proposed several metrics to gauge knowledge concentration [15], [24]. Although they have a similar purpose, these metrics have been referred to with multiple names (*e.g.,* Truck Factor [42], Bus Factor [9], Pony Factor [32]) and computed by aggregating different metrics of code contribution as, for example for each developer, the number of commits in a given module [4], the proportion of lines of code attributed to them in the system [13], and how long ago the developer interacted with a file or module [12]. The Pony Factor (PF) [32], in particular, represents the minimum number of developers who collectively "own" more than half of all the contributions to a project component.

Leveraging a large-scale repository mining framework, Orrei *et al.* [32] computed the PF for 1,011 popular GitHub projects, and found that the majority of the projects in their sample had a PF of one or two, while less than 10 projects had a PF larger than 24. However, given the authors' concerning findings on the prevalence of small PF values in popular projects and the inherently complex nature of software as a system of systems, *could one expect to find larger PF values in the subsystems and modules of large codebases?*

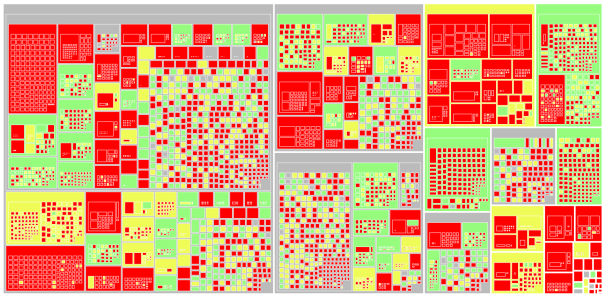

Fig. 1. Pony Factor of the Python programming language repository at files and modules level (PF of 1 in red, 2 in yellow, between 3 and 5 in green, greater than 5 in gray).

Figure 1 shows a glimpse of the complexity behind authorship concentration in files and submodules of a large software codebase (*i.e.,* the Python programming language repository), which remains hidden when focusing only on metrics at a project level. A high project PF can overshadow a reality in which a single author maintains a critical module, creating a single point of failure. Conversely, a low PF might be acceptable (or even expected) for very recent files and submodules, depending on how vital they are for the system.

We replicate the study of Orrei *et al.* [32], examining how the line-level PF is distributed across files, submodules, and projects in a sample of 80 large Free/Libre and Open Source Software (FLOSS) codebases. Our stratified sample includes 72 GitHub projects of varying ages, as well as 8 classic FLOSS projects created before the rise of GitHub.

Analyzing authorship concentration at a more fine-grained scale, in individual system components and for single files at line-level, has implications for the health and maintainability of software projects. As we group PF values into four authorship-concentration levels, we enrich our replication by examining whether and how files and modules progress through these levels over the course of project evolution.

Our results confirm **the prevalence of authorship concentration across the majority of files and directories in large FLOSS projects**, as well as its multifaceted nature, with larger and older modules having a stronger tendency to naturally (or purposefully) progress toward a collective authorship model. Ultimately, our replication elicits fine-grained insights into what researchers and practitioners may expect to be the "norm" for authorship concentration in FLOSS projects, to further investigate divergences and their motivation.

II. Background and Related Work

The concepts of *authorship* and *ownership* are often used interchangeably [24], [33]. To clarify the meaning in our work, we distinguish between them and refer to the notion of code ownership as the individual's or organization's responsibility for implementing and maintaining a given software component [39]. We refer to authorship when considering the individuals who made a given contribution to the codebase [33]. This distinction implies relevant consequences in the case of large FLOSS projects where, for example, the maintainers currently responsible for a given file or module[1] dedicate themselves to reviewing changes rather than authoring changes [15], [39].

Similarly, we distinguish between *authorship concentration* and *knowledge concentration*. Besides introducing review bottlenecks [37], knowledge concentration is the main factor that makes developer turnover a threat to a project's longevity. Excessive knowledge concentration can cause a team to lose a significant portion of knowledge about the project's design choices and implementation details even when a small number of key developers leave [30], [35], [36]. Knowledge concentration can be measured by analyzing development-related artifacts, such as version control system (VCS) logs and records of code changes and code review discussions [21], [28]. In particular, approaches that leverage VCS history assume that developer knowledge of a part of the system is proportional to the number and extent of contributions to the files in that part [12], [13], [28]. *Authorship concentration* is related to the number of developers who wrote a file or module. It is connected to knowledge concentration, but it ignores developer activities other than code writing, such as code reviews and project management duties. We argue that authorship concentration, by itself, provides valuable insights into collaboration (or lack thereof) among co-authors in large software projects and its impact on software maintenance.

A. Experts, Trucks, and Faults

Recommending experts based on the number of contributions authors make to system files has been extensively studied. In 1998, based on an ethnographic case study, Ackerman and Halverson [1] suggested that the inherent difficulty of information reuse made the time since last interaction a good candidate metric for identifying expertise.

Mockus and Herbsleb [28] proposed a graphical tool, the Expertise Browser, that used authorship information to attribute developers' "Experience Atoms".

[1]We use 'folder', 'module', and 'directory', interchangeably.

Fritz *et al.* [12] explored complementing authorship metrics with file interaction data to compose a broader account of developer knowledge, proposing a Degree of Authorship metric to compute the degree to which a developer contributed to the implementation of a specific file.

Authorship data, as a proxy for developer knowledge concentration, has been used to assess the risk of developer turnover. Zazworka *et al.* [42] proposed a method for computing a project's Truck Factor (TF, *a.k.a.* Bus Factor). Torchiano *et al.* [41] followed up discussing potential upper bounds to the TF based on the project's total number of contributors. Avelino *et al.* [2] proposed using the Degree of Authorship metric of Fritz *et al.* [12] to compute a project's TF. We also note two studies that include information other than authorship metrics to estimate knowledge concentration. Jabrayilzade *et al.* [21] measured the TF by including information on a developer's contribution to the review process for file-level changes, as well as the time they spent in meetings discussing such changes. Haratian *et al.* [17] proposed computing a developer's contribution to a module by weighting file-level contributions by their significance through a dependency graph.

Authorship metrics have also been extensively used to judge fault proneness. Bird *et al.* [4] cross-linked authorship-oriented ownership metrics with the occurrence of pre/post-release faults in Microsoft binaries, finding a strong correlation between a high number of minor owners and the occurrence of faults between modules. When replicating the methodology of Bird *et al.* with seven FLOSS projects, Foucault *et al.* [11] observed no significant correlation between ownership and the occurrence of faults when normalizing their metrics on the size of each studied module. Rahman and Devanbu [33] explored line-based blame authorship for relationships between code ownership and faults. They found that the lines that need fixing are less likely to include contributions from multiple authors. Meneely and Williams [26] found mixed results when looking for empirical metrics to assert Linus' Law. Modeling a repository's history as a network of contributions, the authors find strong correlations between the occurrence of source code faults and the co-editing of a file by different groups of non-collaborating developers.

A few authors explored visual approaches to conveying the developers' knowledge of different components of a codebase. Girba *et al.* [13] proposed the "Ownership Map" visual metaphor, which, for every file in a given system, uses a colored timeline to depict which developer authored the majority of its lines at any point in a given time interval. This approach was extended by Hattori *et al.* [18], [19] with their own fine-grained authorship metric (which considered also synchronous file changes). Cosentino *et al.* [9] and Klimov *et al.* [23] explored visual approaches to portraying projects' bus factors at the file and directory levels. In contrast, Burch *et al.* [7] explored coloring the flow of developers joining and leaving the development of different components of two large FLOSS projects using "developer rivers".

B. Reference Work

In the context of authorship concentration metrics, the work of Orrei *et al.* [32] is particularly noteworthy as, to the best of our knowledge, they are the first to provide an overview of authorship concentration across a sample of over 1,000 projects. Motivated by claims of popular high-ranking techno-gurus threatening to use some "mythological" developer productivity proxy to identify employees to layoff, the authors sought to characterize the nuances of the natural distribution (or concentration) of developer productivity in FLOSS.

Using the SEART GitHub Search tool [10], the authors start from projects created before 2016 and with a minimum number of commits, contributors, stars, and watchers (see Figure 2 for the exact thresholds, which we also use in our data collection process). After excluding small projects, those with submodules in separate git repositories, and projects with corrupted git histories imported from other version control systems, the study includes 1,011 repositories totaling over 300 million lines of code.

To address the goal of "metrifying" developer contributions and their distribution within a project, Orrei *et al.* analyze different variants of an authorship concentration metric that balances configurability and ease of computation: the **Pony Factor (PF)**. This metric is attributed to Daniel Gruno,[2] from the Apache Software Foundation,[3] and it is defined as the smallest number of contributors who add up to at least 50% of a project's total contributions. More formally, given f a file, module, or project; D_f the set of developers who contributed to f; $c(d, f)$ the contribution of developer $d \in D_f$ to f; and for $S \subseteq D_f,\ C(S, f) = \sum_{d \in S} c(d, f)$; The Pony Factor $PF(f)$ is defined as follows:

$$PF(f) = \min |S| \ s.t.\ C(S, f) \geq 0.5 \times C(D_f, f) \quad (1)$$

Note that the definition of the developer contribution function $c(d, f)$ is purposely left agnostic with respect to the metric actually used to compute a developer's contribution. Exploring this flexibility in the definition of the PF, Orrei *et al.* compare the distributions of 4 PF variants by switching the function used to gauge developer contribution. Their study, then, compares the projects' PF using *(i)* the number of commits sent by a developer to the file/module; *(ii)* the total number of line changes delivered by a developer to the target file/module; *(iii)* adaptations of *(i)* and *(ii)* considering only the authors who contributed to the project in the last year; and *(iv)* an adaptation of *(ii)* to weight line contributions also according to a memory decay factor, making newer contributions more important than older ones. Testing these four implementations of the PF, the authors see a gradual decrease in the average PF of the projects in their sample (implying a corresponding increase in authorship concentration). While technique *(i)* identifies 99.3% of the sampled projects having PF < 25, technique *(ii)* identifies 99.5% of the projects having PF < 12, and technique *(iv)* identifies 99.3% of projects with a PF smaller than 7.

[2]See https://ke4qqq.wordpress.com/2015/02/08/pony-factor-math/

[3]See https://www.apache.org/foundation/leadership

Additionally, the authors conduct a separate investigation to determine whether the contributors identified as ponies were inclined to code in any particular language or whether their contributions were equally important across all programming languages used in the project. They find that "some crucial developers provide their contributions only in specific locations of the system", and that "if one does not make a distinction [e.g., *by file type responsibility*], there is a substantial risk of underestimating those contributors' importance". It is this particular finding that guides the objectives of our replication study, which seeks to understand the distribution of ponies across the modules and files of each analyzed project.

III. Study Design and Data

Our replication focuses on the distribution of authorship via the PF. Therefore, we reflect on how to interpret the metric as a proxy for how different authors collaborate when writing a project or a part of it. To that end, we extend our replication by classifying the PF of any given file, module, or project into one of four classes of authorship concentration by adapting the definitions of two notable past works [17], [31].

Nordberg [31] discussed the challenges of managing code ownership in an evolving project, and proposed four phases a codebase goes through during its evolution: "*Inception*", when an idea is first conceived; "*Elaboration*", when a sole architect starts implementing the new functionality; "*Construction*", when a team of developers follows the chief architect while collectively constructing upon the initial implementation; and "*Transition*", when the new functionality shifts toward an even more decentralized subsystem-oriented ownership model.

When studying the impact of weighing the bus (truck) factor based on file-significance, Haratian *et al.* [17] compared the results of their approach on subfolders with three different ranges of authorship concentration: Those with a bus factor of one (at a "high risk" of being stalled by turnover); those with a bus factor of two or three ("moderate risk"); and those with a bus factor larger than three ("low risk").

We interpret the state of authorship concentration for the entities (files, submodules, and projects) in our sample in the following four categories of PF. We can have **Hoarded** entities with a **PF of 1**, primarily authored by a single contributor. **Transitioning** entities have a **PF of 2**, meaning two authors sum up to at least 50% of the lines in the current version of the code. These entities can be in a stable state, currently maintained by two developers, in transition from a sole author to another, or in transition from a hoarded state to a collectively owned one. **Team-authored** entities have a **PF between 3 and 5**, and encompass the parts of a system with a lower risk of being stalled by developer turnover, corresponding to the "Construction" stage of Nordberg's model [31]. Finally, **community-authored** projects, submodules, and files have a **PF greater than 5** and correspond to parts of a system that have reached a stable phase of collective maintenance, with no apparent "owner" or team of authors responsible for patching them. These files and modules are at the most negligible risk of being stalled or abandoned by developer turnover.

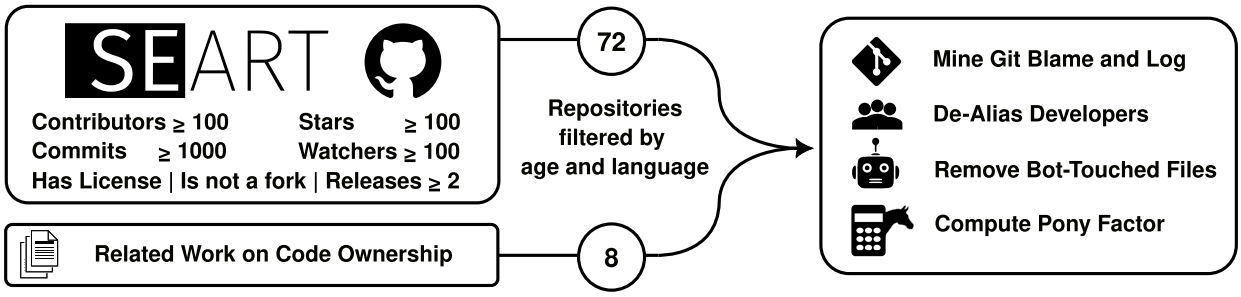

Fig. 2. Overview of our data collection and processing, adapted from that of Orrei *et al.*.

A. Research Questions

Although our reference study does not include any explicit research questions, we organize our replication study and extend the reference one based on four research questions:

RQ1) *What are the PF values of the files, modules, and projects in our sample?* As a fundamental dimension of our replication study, we seek to validate whether the projects in our sample also present a highly skewed PF distribution akin to that found by our reference study. Adding to the replication, we provide a novel extension by conducting a preliminary examination into the distribution of the PF within the files and submodules of each project in our new dataset. Following Brooks' [6] notion of software as systems of systems, and testing the limits of Raymond's [34] Bazaar analogy for FLOSS projects, we include the PF distribution within the components of our software projects.

RQ2) *What local factors affect the PF of files and modules?* Studying authorship concentration at such a fine granularity allows us to pose and answer original research questions that explore whether local factors correlate with the evolution of authorship concentration on a node (file or module/folder). First, we investigate how the progression of the PF relates to general indicators of developer activity on a node, including its size, age, total number of contributions, the number of contributors, and the time without modifications.

RQ3) *To what extent is the PF of a file or module influenced by the project's hierarchy of contents?* We also aim to understand how authorship concentration varies with the logical organization of a codebase. While estimating file and submodule significance usually depends on language-specific approaches, such as identifying dependency graphs, past authors have also explored how a codebase's organization into subdirectories reflects logical proximity, dependence, and specificity [15], [16]. One might expect that files closer to the root directory of a project are more relevant to other submodules or to the project as a whole and are more likely to transition to a collective state of authorship. On the other hand, a file deep in the project's hierarchy might be less important to the project's long-term evolution and is more likely to be hoarded by a single developer. Therefore, in addition to how the PF is affected by local characteristics, we investigate whether the PF depends on a file or directory's position within a project's logical organization.

RQ4) *To what extent is the distribution of the PF influenced by project-specific factors?* Although the model proposed by Nordberg [31] frames ownership progression within subsystems maintained by a single team of developers, project-specific factors could influence the authorship progression of a codebase. Do projects written in different languages manifest different levels of authorship concentration? Are long-lived projects more prone to having authorship concentrated on the few maintainers with sufficient permissions, or do they tend to have their submodules shift toward collective authorship?

B. Data Collection and Processing

Our data collection approach closely matches that used by our reference study and is depicted in Figure 2. It consists of two main phases: one for selecting the FLOSS projects that will constitute our sample dataset, and another for processing the data collected from each repository. Since we compute the PF at a finer granularity than Orrei *et al.*, we cannot base our study solely on their replication package.

1) Project Selection: Since our replication lays special interest in studying how factors such as project language and age affect the distribution of PFs in large FLOSS projects, we must strive to compose a sample that is simultaneously (*i)* specific to projects with a large number of lines of code; and (*ii)* diverse in terms of age and language [29]. Existing software repository querying techniques are usually restricted to a single git forge (*e.g.,* GitHub) and rarely account for some of the oldest FLOSS projects still actively maintained (*e.g.,* Linux kernel, Chromium, GIMP) [10]. Focusing on a select pool of hand-picked FLOSS projects could lead to a biased sample. We balance the limitations of both project selection approaches by adopting them in a complementary manner.

First, we follow our reference work and use the SEART GitHub Search [10] utility as a primary tool for selecting suitable projects. The tool allows for querying repositories hosted on GitHub with project specific characteristics, such as size, language, and project "popularity". We also refer to the values used in the reference work when filtering projects based on minimum thresholds for the number of commits (1,000), contributors (100), stargazers (100), watchers (100), and releases (2) in the projects' histories. These thresholds aim at obtaining projects that have a non-trivial development history, a plurality of contributors, relevance for the community, and multiple release candidates.

Our initial query results in 3,233 candidate repositories, all created on or after 2008. As we are interested in studying the influence of project language and age, we stratify our sample to ensure approximately equivalent numbers of projects across different languages and creation dates. We split the projects into four bins according to the year they were created in: 2025–2021, 2020–2016, 2015–2011, and before 2011. Next, we select the six most used languages in our sample, ensuring that at least three projects use each language in each of the four age bins. In our case, these languages are C, C++, JavaScript, TypeScript, Java, and Python. Then, for each combination of age-bin and language, we select the three projects with the most lines of code (LoC), again prioritizing our sample's specificity toward large projects. This filtering yields a partial dataset of 72 GitHub projects. Among them are notable FLOSS libraries and frameworks, such as the Numpy, Pytorch, and Pandas Python libraries, and the Spring Java framework for building web servers.

We follow previous work to identify large, long-lived FLOSS projects that have been used as case studies on code ownership. Table I presents age, size, and references to the related work they were studied in, for the 8 projects we use to complement the sample that originated from GitHub Search. Since we only select three projects per age bin from GitHub, these 8 older FLOSS projects make up a sizable portion of our sample and help maintain a roughly equal distribution of projects by age. Regardless of this limitation in the number of projects, the total number of lines of code in our sample (806M LoC) is over double that of our reference work (342M LoC). We share both the scripts used for project filtering and the complete list of **80 projects** in our replication package.

TABLE I
NOTABLE FLOSS PROJECTS TAKEN FROM PREVIOUS WORKS.

Project	Age (years)	No. Lines	Used by
Chromium	18	22M	[30], [35]
Firefox	24	63M	[16], [36]
GIMP	30	5M	[20], [30]
Apache HTTP	31	1M	[22], [27]
Python	35	3M	[5], [7]
Linux kernel	35	38M	[27], [30]
GCC	39	24M	[22], [27]
Emacs	40	5M	[20]

2) Data Processing: Since the replication package for our reference work does not include the scripts used for data collection, we reimplement them based on the authors' descriptions. For the sake of reproducibility, we include the scripts used to collect and process the git repositories, as well as the git hash of the commit from which we extracted the data for each project, in our replication package.

For every project in our sample, we first clone its corresponding git repository. We choose not to follow the cloning process recursively and stop if a given submodule is stored at a different git repository. We exclude assets with common file extensions pertaining to images, video, or other binary files that are not adequately versioned by git's line blame feature.

Our data processing pipeline includes Python and Bash scripts. Since git-blame and git-log are traditionally implemented as single-threaded command line tools, we use GNU Parallel [38] to take advantage of our multi-core infrastructure.

While parsing through the data, we remove author aliases following the approach described by our reference work. We consider two identities (*i.e.,* name-email pairs) to refer to the same developer if (*i)* the candidate names have a Levenshtein similarity higher than 90%; or (*ii)* the candidate emails have a Jaro-Winkler similarity of over 90% while the candidate names have a Levenshtein similarity higher than 70%.

File level PF: For every file, we collect its size in number of lines, the date of the first commit still "visible" in the current blame of the file, the date of the most recent commit made to it, the number of total commits made to the file, how many (de-aliased) developers contributed to the current version of the file (as listed by git-blame), and the line-level blame PF.

We use Equation (1) with $c(d, f)$ being the number of lines attributed to developer d on the current version of file f, and $C(D_f, f)$ being the total file size in lines.

We exclude empty files and files with only one commit from our file-level analyses. We adopt the line-level blame for our PF calculation because, as shown in Section II, it offers a reasonable perspective into the current authorship concentration of a system at a relatively low computational cost and it has been extensively used to measure authorship in the past [3], [9], [13], [16], [33].

We exclude any file that has a contribution by a *bot* listed in the git blame at the analyzed commit. We identify CI/CD bots containing the substring `[bot]` in the author's name, once again following the approach of our reference work [32]. Our first sample of 72 GitHub projects included five projects that had more than 25% of their files discarded because they were altered by bots. Therefore, we replace each of these five projects with the next longest project in terms of LoC from the same age bin and main language (see Section III-B1). For the final sample, we compute the percentage of files discarded on a project basis and confirm that only a minority of the studied files had contributions from bots (13.5k corresponding to 0.7% of all files). We aggregate file metrics recursively to compute directory-level metrics.

Directory level PF: We consider the directory size to be the sum of the lines in the directory's contents. The first interaction date of a directory is the date of any commit that contributed to a line that is still unchanged in the current version of the files in the directory. Similarly, the date of the most recent commit is considered to be the most recent of any file in the directory. We do not count the number of commits done in the directory, but we consider the overall number of contributors listed in the blame of the directory's files.

For calculating the PF of a directory f, we adapt Equation (1) to consider $c(d, f)$ to be the sum of the number of lines developer d has attributed to them on the current version of every file (or subdirectory, recursively) contained in f, and $C(D_f, f)$ to be the sum of the length of the directory's contents in number of lines.

As shown in Table II, our project selection process yields a diverse sample, balanced by age and language (12–17 projects per language). On the other hand, all projects have between 213k and 63M LoC, ensuring our goal to capture large FLOSS projects in our replication study.

TABLE II
DESCRIPTIVE STATISTICS OF THE 80 PROJECTS IN OUR SAMPLE.

Variable	Min	Q1	Median	Q3	Max
Creation date	Sep'86	Dec'07	Dec'11	Aug'17	Jun'24
# of contributors	84	213	465	951	29,108
LoCs	213k	2.4M	6.7M	14M	63M

In Table III, we present descriptive statistics for our variables related to files and modules. We mark with f the variables computed only for files, and d the ones computed only for directories. Our projects total over 806M LoC distributed across close to 400k folders containing 1.8M files.

TABLE III
DESCRIPTIVE STATISTICS FOR THE FILES AND MODULES IN OUR SAMPLE.

Variable	Min	Q1	Median	Q3	Max
Node size (lines)	1	20	53	177	63M
# of contributors	1	1	2	3	29k
# of commits - f	2	3	4	9	79k
First commit date	Sep'86	Feb'18	Sep'21	May'24	Feb'26
Last commit date	Jan'93	Aug'21	Jul'24	Aug'25	Feb'26
Node depth	0	5	6	8	22
Dir. height - d	1	1	1	2	22
# of children - d	1	1	3	6	9k

IV. RESULTS

We follow our study's fourfold objectives by reporting and discussing the results of each of our research questions separately. We reserve Section V for a broader discussion on the implications of the results from all our RQs.

A. (RQ1) What are the PF values of the files, submodules, and projects in our sample?

First, we present the PF values for the 80 projects in our sample. As seen in Figure 3, the histogram of the PF values we found strongly resembles that presented by Orrei *et al.*, with a clear prevalence of projects with low Pony Factor values.

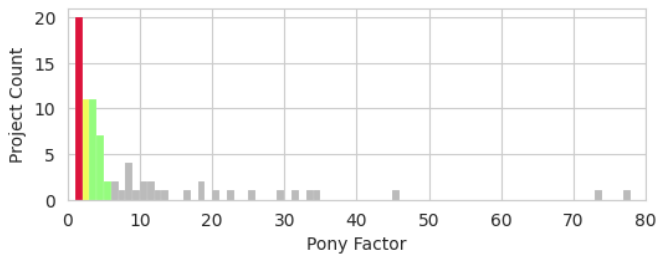

Fig. 3. Histogram of Pony Factor values for the projects in our sample. We removed two outliers from the histogram: the Chromium (PF = 263) and Linux kernel (PF = 360) projects.

We note that 70 (87.50%) projects in our sample of large FLOSS codebases have a Pony Factor smaller than 25, and 42 (52.50%) have a Pony Factor of up to 3. If we consider the files and modules of these projects, we see an even starker concentration of contributor authorship.

In our sample, 1.5M files (87.12%) have a PF of 1, as do 328k (82.17%) folders. The second most frequent class of files and modules is the one transitioning between hoarded and collective authorship models. There are 196k (10.81%) files, and 50k (12.60%) directories have a PF of 2. A small number of nodes are collectively authored: 34,900 files (1.92%) and 17,700 folders (4.45%) have a PF between 3 and 5. Moreover, only 2.8k files (0.16%) and 3.1k folders (0.78%) have a PF greater than 5, hinting at the rarity of community-authorship.

Table IV presents the distribution of files and directories across our four PF classes in the sampled projects. There is a clear prevalence of files and folders with PF = 1 across all our projects. Over half of the projects have at least 90% of hoarded files. The project with the smallest proportion of hoarded files has over 63% of files with PF = 1. The scarcity of nodes with PF $>$ 5 is equally noteworthy. Over half of the projects do not have a single community-authored file, and at least 25% of all projects do not have any community-authored submodules. For the projects that do have community-authored nodes, they make a minuscule proportion of all files (Q3: 0.08%, max: 1.02%) and submodules (Q3: 0.80%, max: 7.00%).

TABLE IV
DISTRIBUTION OF PF BIN PROPORTIONS ACROSS OUR 80 PROJECTS.

Node	PF	Min	Q1	Median	Q3	Max
Files	1	63.59%	82.38%	90.13%	93.19%	99.77%
Files	2	0.23%	6.62%	9.26%	15.84%	26.39%
Files	$3-5$	0.00%	0.25%	0.88%	2.06%	11.36%
Files	≥ 6	0.00%	0.00%	0.00%	0.08%	1.02%
Dirs	1	48.79%	76.53%	84.25%	90.86%	99.10%
Dirs	2	0.91%	7.79%	10.62%	17.71%	26.49%
Dirs	$3-5$	0.00%	0.77%	2.10%	5.73%	19.76%
Dirs	≥ 6	0.00%	0.00%	0.13%	0.80%	7.00%

(RQ1) Discussion: We acknowledge that the distribution of PF values across the projects in our sample is not as strongly skewed towards lower values as that reported by Orrei *et al.* [32]. While the authors of the reference work reported that over 99% of the projects in their sample had a PF smaller than 25 when considering their commit-level implementation of the Pony Factor, that proportion shrinks to 87.50% in our sample. This difference does not contradict the findings of the reference work, as our implementation of the Pony Factor is slightly different (using line-level blame data, instead of the number of commits written by each contributor), and the projects in our sample are, on average, larger and older than those used by Orrei *et al.*. On the contrary, we argue that our results corroborate the reference work by evidencing a tendency towards authorship concentration in the large projects we studied, with over half of these projects having up to three authors who wrote the majority of their code.

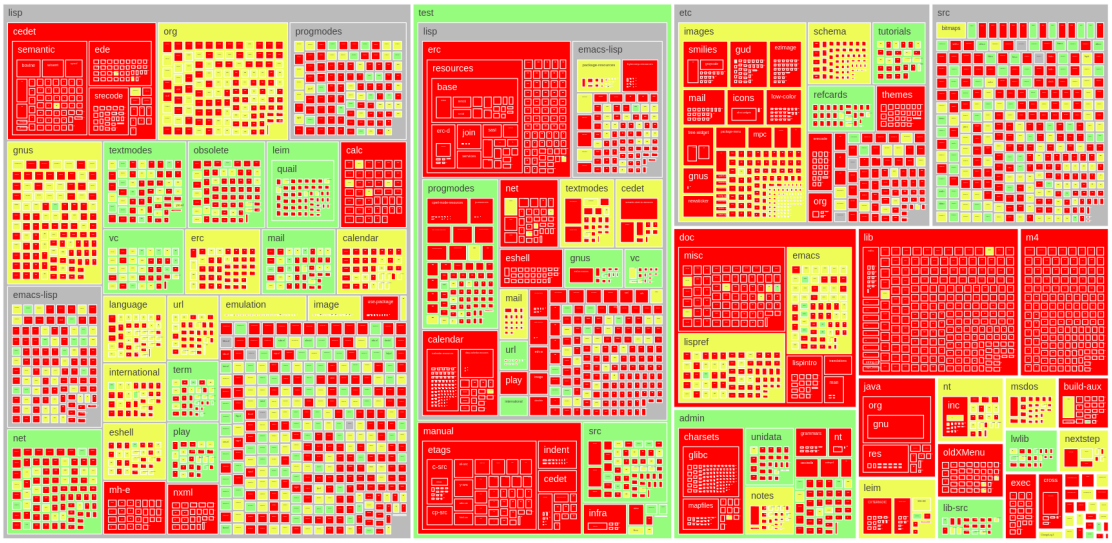

Fig. 4. Pony Factor of the Emacs repository at files and modules level (PF of 1 in red, 2 in yellow, between 3 and 5 in green, greater than 5 in gray).

Figure 4 depicts the PF-based authorship classes for the files and modules of the Emacs text editor. The project serves as a clear example of the insights and implications overshadowed by analyzing a project with a single project-wise PF value.

Considering all lines of code in the project, Emacs has a PF of 9, a value that could be interpreted as the sign of a collaborative, turnover-resistant FLOSS project. Looking at the PF of specific modules of the Emacs project, however, paints a more nuanced picture of how collaboratively its code was authored. Core modules, such as the Lisp and C implementations of the editor's source code, present a high PF and indicate a community-authored effort, as seen in the `lisp` (left) and `src` (upper right corner) sections of the treemap. Conversely, in the lower right sector of the plot we see modules with a PF of 1, revealing specific parts of the project that are at a high risk of being stalled by developer turnover, including the Java port of Emacs (`java` directory), most of the official documentation on the project's manual page (`doc`), and the configuration scripts used to build and install the editor (`m4`).

While a deeper discussion on the importance of the different submodules of the Emacs project would be guesswork without the insights of an actual maintainer of the project, we can explore general tendencies among the modules with different levels of authorship concentration. Are the modules with the highest authorship concentration younger, or do they depend on niche knowledge of the project's inner workings? As we follow the next three RQs throughout our analyses, we piece together what local and project-wide factors accompany significant changes in authorship concentration within a project.

(RQ1) PF Distribution: Key Results

We replicate Orrei *et al.*'s findings by observing that, even on the large, long-lived projects in our sample, the majority of codebases (52.50%) are attributed to up to three contributors. The vast majority of a codebase's files and submodules are authored mainly by a single developer.

B. (RQ2) What local factors affect the PF of a file or module?

We follow our study by investigating how local metrics relate to authorship concentration, measuring Spearman's correlation coefficient between the PF and each local variable listed in Table III. All reported correlations are statistically significant ($p < 0.05$ adjusted using Bonferroni Correction), yet only the strongest coefficient reveals a moderately strong correlation. Perhaps unsurprisingly, the number of contributors whose contributions remain in the file or module exhibits the strongest correlation with the PF (0.56 for folders, 0.47 for files), followed by the total number of commits involving a given file (0.35). The date of the last contribution received has the third strongest correlation to the PF for files (0.18) and directories (0.32). These correlations indicate a weak to moderate tendency for recently modified nodes to be collectively owned, and for nodes not actively maintained to exhibit stronger authorship concentration. The size of the files or modules shows the second-lowest correlation coefficient with the PF: 0.15 for files and 0.28 for directories.

The weakest correlation coefficients in our set of variables belong to the date of the first commit still listed in the blame of files (-0.13) and folders (-0.15). Nevertheless, and as seen in Figure 5, even the variable with the weakest correlation with the PF paints a different picture when we consider our four classes of authorship concentration described in Section III.

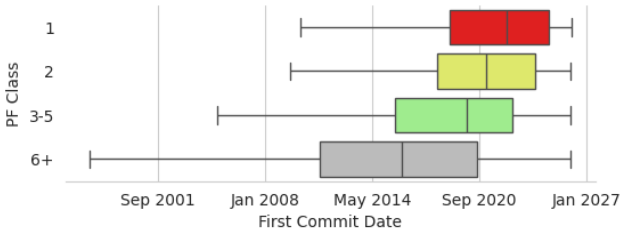

Fig. 5. Distribution of folders by first commit date.

Figure 5 presents the distribution of folders by the date of first commit, as reported in their content's blame, ordered by PF bin. Mann-Whitney U tests with p-values $<< 0.0001$ confirm that the number of folders by age across the different PF bins does not follow a single random distribution. After computing Cliff's delta for every pair of PF bins, we observe medium to large effect sizes between community-authored (in gray) folders against hoarded (-0.60, in red), transitioning (-0.47, in yellow), and team-authored (-0.32, in green) ones. The median age for the folders in the different bins is 3 years old for PF = 1; 5 years old for PF = 2; 6 years old for PF between 3 and 5; and 10 years old for PF ≥ 6.

Similarly, for time since last modification, Mann-Whitney U tests confirm that the values follow different distributions in each bin. Every pair of distributions is separated by a moderate or strong Cliff's delta effect size. A Cliff's delta of 0.80 between hoarded and community-authored folders reveals how collectively authored modules are less likely to experience long periods without maintenance.

When looking at abandoned folders, the difference between hoarded and collectively authored folders is starker. While the ratio of folders with a PF of 1 is 82.17%, that same ratio jumps to 93.11% and 97.24% when considering folders abandoned after 1 and 5 years without modification, respectively. Out of 3.1k collectively-authored folders, only 27 (0.09%) and 10 (0.03%) folders are considered abandoned, following the two previous abandonment thresholds.

(RQ2) Discussion: Our results for RQ2 confirm that authorship concentration shows no substantial relationship with local metrics, such as node size or age. The weak correlation between file size and PF shows that larger subsystems are not commonly composed of comparably sized contributions from different authors, but are authored by a concentrated sample of authors that tends to be only slightly larger than that of simpler system components.

Perhaps more interestingly, however, is how our PF-based authorship concentration classes relate to the time since the last and first modifications. The positive correlation between time of last contribution and PF reveals how abandoned nodes tend to be hoarded, suggesting that these nodes might be at a higher risk of developer turnover. Conversely, the negative correlation between node creation date and PF reflects the scarcity of newly created community-authored files.

(RQ2) Local Metrics: Key Results
The PF exhibits only weak to moderate correlations with local metrics, including file/module size, number of contributors, and contributions. When comparing nodes across different authorship concentration models, our results suggest that a node's progression toward a community-authored state, if naturally occurring, can be a slow and decade-long process.

C. (RQ3) To what extent is the PF of a file or module influenced by the project's hierarchy of contents?

We first explore whether there exists any notable Spearman correlation between the PF and the variables from Table III pertaining to the repository's internal file hierarchy. These variables are 1) *Node Depth:* the distance from the root directory in the directed graph that represents the repository file tree; 2) *Directory Height:* the longest path between a directory and a leaf it contains in such a graph; and 3) the number of immediate children of a directory.

Although all statistically significant, the correlations are weaker than those associated with the local metrics. The strongest correlation to the PF comes with the number of immediate children of submodules (0.20), followed by the directory's content height (0.15). The node depth presents the weakest correlation with the PF of files (-0.11) and submodules (-0.09). Comparing how these metrics vary for nodes from different authorship concentration classes reveals statistically significant differences, as once again asserted using Mann-Whitney U tests yielding p-values $<< 0.0001$.

We observe large and moderate Cliff's delta effect sizes between hoarded (-0.59), transitioning (-0.45), and team-authored (-0.27) files when compared to community-authored ones. The equivalent effect sizes for folders are comparable to those seen for files (-0.54, -0.46, and -0.37, respectively) and suggest a tendency for community-authored nodes being closer to the project's root directory. The same tendency is corroborated by analyzing Cliff's delta for the distribution of directory height by PF bin. Notably, we observe moderate and large effect sizes for hoarded (0.46) and transitioning (0.34) folders compared with community-authored ones.

(RQ3) Discussion: Considering both directory height and the number of immediate children as proxies for complexity, the correlation coefficients suggest only a slight tendency for more complex modules to be more commonly collectively authored. Notably, this tendency closely aligns with the correlation between PF and node size in lines discussed in RQ2.

Nevertheless, the significant differences in node depth, folder height, and the number of contents across PF bins reveal that these factors relate to which parts of a codebase are more commonly authored collectively. We see submodules of greater complexity or greater overall importance to the project often adopt a more collaborative authorship model. This tendency is supported by the notable effect sizes found with Cliff's delta and the positive correlation between directory height and the number of contents against the modules' PF. Similarly, the negative correlation between PF and node depth indicates that collectively-authored nodes tend to be less specific to niche parts of the codebase, or closer to the root of the repository.

(RQ3) Hierarchy of Project Contents: Key Results
The PF exhibits only weak correlations with node depth and with the height or number of contents in directories. Nevertheless, authorship concentration is lighter for nodes close to the project's root, as well as for folders containing a larger number of nodes or with a deep sub-hierarchy.

D. (RQ4) To what extent is the distribution of the PF influenced by project-specific factors?

When comparing the distribution of nodes across PF bins by project language, our Mann-Whitney U tests yield p-values greater than 0.05, so our sample provides no empirical evidence that the project's primary language affects whether a project is more or less susceptible to authorship concentration.

Investigating the relationship between the distribution of nodes by PF, project age, number of files, and number of contributors, we see moderate to strong correlations emerge. The association between the proportion of collectively-owned nodes and the overall number of contributors is noteworthy.

Figure 6 depicts the proportion of community-authored files ordered by overall contributors, with a Spearman correlation coefficient of 0.80. For folders, the correlation coefficient with the project's number of contributors is 0.66. Besides confirming that project language has no impact on authorship concentration, we see a tendency for projects authored for a wider audience to have more files authored collaboratively.

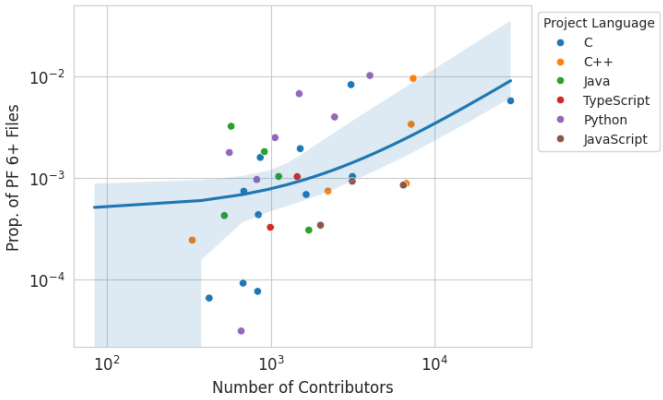

Fig. 6. Log-log plot with the proportion of files with PF ≥ 6 by number of contributors.

We note moderate correlations between project-level metrics and the proportions of nodes in different authorship concentration states. Project age has a negative correlation with the proportions of hoarded files (-0.35) and folders (-0.30), but a positive correlation with the proportions of team-authored files (0.36) and folders (0.37), as well as with community-authored nodes (files 0.36, directories 0.38). A project's number of files shows a positive correlation to the proportions of community-authored files (0.45) and directories (0.30).

(RQ4) Discussion: Our analyses on the influence of project-level metrics reveal insights into the organic proportions of files and modules split across different authorship concentration models in FLOSS. We see that the strong prevalence of modules with low pony factors, evidenced in RQ1, has no clear connection to the project languages. More broadly, the finding that the project-level correlations are usually stronger than local metrics hints at the idea that project-wide decisions about how developers co-author code are better indicators of how authorship concentrates within a project.

Notably, the correlation between project age and the distribution of collaboratively-authored modules is higher than that between the individual ages of directories' contents and their PFs. This suggests that, as a project matures, its development model shifts to a state that favors the development of collectively authored submodules.

(RQ4) Project Level Metrics: Key Results
Project-level metrics show a stronger association with overall PF distribution than local metrics do with the PF of individual nodes. This suggests that project maturity drives a global organizational shift, evolving to a development model that inherently favors collectively authored submodules over concentrated authorship.

V. General Discussion

From a technical standpoint, we remark no significant difficulties replicating the reference work of Orrei *et al.* [32]. Although the replication package associated with it did not include the scripts used to mine the software repositories, the reference study provided clear descriptions of the methodology adopted. Since we replicate the reference work under slightly different conditions (on a sample of large projects, and with adjustments to how we computed the PF), we see our results corroborating the strong prevalence of low PF values found by Orrei *et al.* Together with the results of RQ1, RQ4 shows that this category of projects thrives even when over half of the codebase's subsystems have a single author.

Our file- and module-level results also align with the significant authorship concentration observed by other previous studies, such as those of Greiler *et al.* [14]. In the set of 130 GitHub repositories studied by Avelino *et al.* [2], 65% of the projects had an overall truck factor of 1 or 2. Therefore, setting community-authorship as a goal for the majority of a project's modules might be equally unreasonable and unnecessary. While the finding that collectively authored modules are less likely to be abandoned than hoarded modules leaves no doubt about the long-term benefits of collective authorship, the sheer prevalence of hoarded modules in our sample of large FLOSS projects hints at the challenges connected to having a prevalence of team- and community-authored subsystems.

The variety of factors involved in software maintenance and evolution, coupled with the weak correlation coefficients obtained in RQ2 and RQ3, proves how the evolution of authorship (and, to some extent, knowledge [21], [30]) in a codebase is a nuanced phenomenon. Neither larger nor older modules show a clear tendency to be collaboratively authored, suggesting that the factors that actually guide the development of a module toward a more collaborative and decentralized approach are complex and depend on both code characteristics and exogenous factors that shape how developers organize themselves around the codebase. Certain development policies favor authorship concentration by imposing the burden of a heavily centralized patch review process [37], while looser reviewing requirements favor the long-term spread (the opposite of concentration) of authorship and knowledge [15].

Implications for Practitioners: The overall proportion of nodes we found at different authorship concentration levels and varying creation dates can give rise to recommendations for practitioners. Stimulating collective ownership practices from the early stages of a file or submodule's development might speed up the process of authorship dilution, with more participation of project maintainers and other developers, which can be slow when it happens organically and autonomously in FLOSS projects. Additionally, teams willing to avoid the risk of developer turnover should proceed with caution when a codebase's ratio of hoarded submodules exceeds 90%: They should prioritize knowledge-sharing practices, especially for hoarded modules written longer ago or with greater proximity to the repository's root.

Implications for Researchers: Besides the scripts and data in our replication package, our results validate previous studies (in particular that of Orrei *et al.* [32], by replication) and extend them for future researchers with a broad ground truth of the statistics of authorship concentration at varying levels in FLOSS projects. The broad view of our analyses reveals generally weak correlations between PF distribution and local metrics, hinting at more intricacies that must be investigated to understand why specific projects and submodules evolve into a collective state of authorship. Future work might explore how other metrics (*e.g.,* code churn, commit frequency) relate to the PF, and examine how the choice of alternative metrics for computing the PF, or equivalent metrics, affects the results on FLOSS authorship concentration.

VI. Threats to validity

1) Construct Validity: We acknowledge that our study inherits limitations from its reference study. In particular, we note that misclassifications from the approaches used for identifying developer aliases and bots can affect our estimations of the Pony Factor. Moreover, the metric used for measuring developer contributions also comes with its own limitations. Previous authors have explored the effects of using different metrics from version control systems to measure authorship and experience, some of which explored limitations of git's traditional blame metric, which can fail to track authorship for contributions shorter than a line [8], [25], [27].

2) Internal Validity: We mitigate our replication's susceptibility to selection biases by deliberately limiting our project selection process to ensure a balanced ratio of projects written in different languages and age groups. We also note that survivorship bias could have affected our project selection if we had included projects that are no longer maintained (which we did not). Furthermore, we acknowledge that the correlation comparisons made in the RQs that extend our replication study could have been affected by confounding factors, as one local metric could be strongly related to others. An analysis of the Variance Inflation Factor (VIF) revealed that this should not be the case, as the largest VIF value we observed (2.29, related to the number of contributors on a node) suggests only a weak to moderate correlation with other variables.

3) External Validity: We recognize that the results of our replication may not reflect how authorship concentration evolves in smaller FLOSS projects or in proprietary software in general. However, our reasonably sized sample can provide satisfactory generalizability to other large FLOSS projects.

VII. Conclusion

Besides replicating the investigation of Orrei *et al.* [32] on the Pony Factor, we sought to examine how authorship concentration evolves at a finer granularity and what factors influence its variation within and between software codebases. We conducted and presented a replication of their study while also investigating the factors influencing authorship concentration at the file, module, and project levels in 80 popular and large FLOSS codebases. Ultimately, our investigation complemented our reference work by exploring whether Eric Raymond's conception of a Bazaar as a lively, chaotic, and plural environment aligns closely with how authorship concentration evolves in reality [34].

In addition to confirming our reference work's results on the prevalence of low Pony Factor values in FLOSS, our data reveals nuances that project-level metrics fail to capture when showing how authorship concentrates in practice. The fact that we observed less authorship concentration in large, mature, and foundational subsystems, while more hoarding of nodes in newly created parts of these complex codebases, implies that active diligence is needed from a project's organization to ensure a system-wide collective authorship policy. Such a stark prevalence of hoarded files and submodules raises the question of what the limits of the Bazaar analogy might be. The results of our study show that the seemingly paradoxical rise of Bazaars from a set of individually built contributions, the Cathedrals, is a consequence of the inherently complex way in which FLOSS grows and evolves organically.

Replication Package

Our replication package includes the scripts used to select the projects in our sample, to mine their PF information, the data used for our replication of the reference study, and the scripts used for the analyses presented in this work. The package can be found at the following url: https://doi.org/10.6084/m9.figshare.31549921

Acknowledgments

This research is supported by the São Paulo Research Foundation — FAPESP (2022/16618-2, 2025/26679-7), and the São Paulo State Data Analysis System Foundation – SEADE (FAPESP 2023/18026-8). The authors would also like to thank the Swiss National Science Foundation (SNSF) for their financial support through the project "FORCE" (SNF Project Number 232141).

References

[1] M. S. Ackerman and C. Halverson, "Considering an organization's memory," in *Proceedings of the 7th ACM Conference on Computer Supported Cooperative Work (CSCW)*. ACM, Nov. 1998, p. 39–48.

[2] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *Proceedings of the 24th International Conference on Program Comprehension (ICPC)*. IEEE, May 2016, pp. 1–10.

[3] G. Avelino, L. Passos, F. Petrillo, and M. T. Valente, "Who can maintain this code?: Assessing the effectiveness of repository-mining techniques for identifying software maintainers," *IEEE Software*, vol. 36, no. 6, p. 34–42, Nov. 2019.

[4] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! Examining the effects of ownership on software quality," in *Proceedings of the 19th SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE)*. ACM, Sep. 2011, pp. 4–14.

[5] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proceedings of the 16th International Symposium on Foundations of software engineering (FSE)*. ACM, Nov. 2008, pp. 24–35.

[6] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1978.

[7] M. Burch, T. Munz, F. Beck, and D. Weiskopf, "Visualizing work processes in software engineering with developer rivers," in *Proceedings of the 3rd Working Conference on Software Visualization (VISSOFT)*. IEEE, Sep. 2015, pp. 116–124.

[8] G. Canfora, L. Cerulo, and M. Di Penta, "Identifying changed source code lines from version repositories," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*. IEEE, May 2007, p. 14–14.

[9] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, Mar. 2015, pp. 499–503.

[10] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *Proceedings of the 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 560–564.

[11] M. Foucault, J.-R. Falleri, and X. Blanc, "Code ownership in open-source software," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, May 2014, pp. 1–9.

[12] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, vol. 1, May 2010, p. 385–394.

[13] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE'05)*, Sep. 2005, pp. 113–122.

[14] M. Greiler, K. Herzig, and J. Czerwonka, "Code ownership and software quality: a replication study," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. Florence, Italy: IEEE Press, May 2015, pp. 2–12. [Online]. Available: https://dl.acm.org/doi/10.5555/2820518.2820522

[15] F. Hajari, S. Malmir, E. Mirsaeedi, and P. C. Rigby, "Factoring expertise, workload, and turnover into code review recommendation," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 884–899, Apr. 2024.

[16] C. Hannebauer, M. Patalas, S. Stünkel, and V. Gruhn, "Automatically recommending code reviewers based on their expertise: an empirical comparison," in *Proceedings of the 31st International Conference on Automated Software Engineering (ASE)*. ACM, Aug. 2016, pp. 99–110.

[17] V. Haratian, M. Evtikhiev, P. Derakhshanfar, E. Tüzün, and V. Kovalenko, "BFSig: Leveraging file significance in bus factor estimation," in *Proceedings of the 31st Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov. 2023, p. 1926–1936.

[18] L. Hattori and M. Lanza, "Mining the history of synchronous changes to refine code ownership," in *Proceedings of the 6th International Working Conference on Mining Software Repositories (MSR)*. IEEE, May 2009, p. 141–150.

[19] L. P. Hattori, M. Lanza, and R. Robbes, "Refining code ownership with synchronous changes," *Empirical Software Engineering*, vol. 17, no. 4, p. 467–499, Aug. 2012.

[20] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. Gonzalez-Barahona, "Using software archaeology to measure knowledge loss in software projects due to developer turnover," in *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS)*. IEEE, Jan. 2009, pp. 1–10, iSSN: 1530-1605.

[21] E. Jabrayilzade, M. Evtikhiev, E. Tüzün, and V. Kovalenko, "Bus factor in practice," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, Oct. 2022, p. 97–106.

[22] H. Kagdi, M. Hammad, and J. I. Maletic, "Who can help me with this source code change?" in *Proceedings of the 24th International Conference on Software Maintenance (ICSM)*. IEEE, Sep. 2008, pp. 157–166, iSSN: 1063-6773.

[23] E. Klimov, M. U. Ahmed, N. Sviridov, P. Derakhshanfar, E. Tüzün, and V. Kovalenko, "Bus factor explorer," in *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, Sep. 2023, pp. 2018–2021.

[24] U. A. Koana, Q. H. Le, S. Rahman, C. Carlson, F. Chew, and M. Nayebi, "Examining ownership models in software teams," *Empirical Software Engineering*, vol. 29, no. 6, p. 155, Sep. 2024.

[25] M. Kondo, D. M. German, Y. Kamei, N. Ubayashi, and O. Mizuno, "An empirical study of token-based micro commits," *Empirical Software Engineering*, vol. 29, no. 6, p. 148, Sep. 2024.

[26] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of linus' law," in *Proceedings of the 16th Conference on Computer and Communications Security (CCS)*. ACM, Nov. 2009, pp. 453–462.

[27] X. Meng, B. P. Miller, W. R. Williams, and A. R. Bernat, "Mining software repositories for accurate authorship," in *Proceedings of the 29th International Conference on Software Maintenance (ICSM)*. IEEE, Sep. 2013, pp. 250–259.

[28] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*. ACM, May 2002, p. 503–512.

[29] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2013, pp. 466–476.

[30] M. Nassif and M. P. Robillard, "Revisiting turnover-induced knowledge loss in software projects," in *Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Sep. 2017, pp. 261–272.

[31] M. Nordberg, "Managing code ownership," *IEEE Software*, vol. 20, no. 2, p. 26–33, Mar. 2003.

[32] V. Orrei, M. Raglianti, C. Nagy, and M. Lanza, "Contribution-based firing of developers?" in *Proceedings of the 31st Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov. 2023, p. 2062–2066.

[33] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. ACM, May 2011, p. 491–500.

[34] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, Sep. 1999.

[35] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "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)*. ACM, May 2016, pp. 1006–1016.

[36] G. Robles and J. M. Gonzalez-Barahona, "Contributor turnover in libre software projects," in *Open Source Systems*, E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, and G. Succi, Eds. Springer US, 2006, p. 273–286.

[37] X. Tan, M. Zhou, and B. Fitzgerald, "Scaling open source communities: an empirical study of the Linux kernel," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*. ACM, Oct. 2020, p. 1222–1234.

[38] O. Tange, "GNU Parallel 20231122 ('Grindavík')." [Online]. Available: https://doi.org/10.5281/zenodo.10199085

[39] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM, May 2016, p. 1039–1050.

[40] P. Thongtanunam and C. Tantithamthavorn, "Code ownership: The principles, differences, and their associations with software quality," in *Proceedings of the 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Oct. 2024, pp. 379–390.

[41] M. Torchiano, F. Ricca, and A. Marchetto, "Is my project's truck factor low? Theoretical and empirical considerations about the truck factor threshold," in *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics (WETSoM)*. ACM, May 2011, p. 12–18.

[42] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider, "Are developers complying with the process: an XP study," in *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Sep. 2010, pp. 1–10.