Exploring Code Ownership Concentration in the Linux Kernel

Arthur Pilone
Institute of Mathematics, Statistics,
and Computer Science (IME-USP),
University of São Paulo (USP)
São Paulo, Brazil
arthurpilone@ime.usp.br

Paulo Meirelles
Institute of Mathematics, Statistics,
and Computer Science (IME-USP),
University of São Paulo (USP)
São Paulo, Brazil
paulormm@ime.usp.br

Abstract

A fundamental aspect determining the design, evolution, and maintenance of a software project is the set of values, practices, and methods its contributors follow throughout its development. In large software projects, practices and policies commonly vary for individual software components (subsystems) or subordinate subprojects. While the project-level organization and context provide a common background for their subprojects, the internal developer hierarchies and the contributions of these subprojects to the parent project give rise to differences in how developer subcommunities coauthor components of large software projects. We conduct a case study on the distribution of ownership concentration metrics across Linux kernel subsystems to explore how subproject-level development models affect process metrics extracted from a common software codebase. We found a statistically significant difference in the distributions of minor and major contributors across subsystems, as well as considerable disparities in the strength of correlations between ownership concentration and local code metrics. Our results reveal how the development model of a single project can vary across its software components and point to the benefits of mining software repositories with awareness of a project's context, domain, and developer organization.

Keywords

Ownership Concentration, Code Ownership, Software Maintenance, Open Source Software

1 Introduction

For a while, the empirical software engineering community has leveraged the popularity and centralization of hosting platforms made for sharing Git repositories and collaborative code development (*e.g.,* GitLab and GitHub) [15]. By querying these platforms, researchers can arrange samples of hundreds or thousands of projects for large-scale studies on the adoption of technologies and practices [20], community engagement [19], and developer behavior [17, 34]. Of particular note are the large-scale studies that mine process metrics and, based on contribution information from these repositories, investigate how teams organize themselves around the code using notions such as expertise and ownership [11, 23, 27].

On the other hand, Conway's Law[1] has long stood as a classic and foundational reminder of the tight interplay between how developers arrange themselves and the logical architecture of the codebase they author [6, 7].

[1] *Organizations, who design systems, are constrained to produce designs which are copies of the communication structures of these organizations.*

While for most software projects the specific arrangement of developer subgroups in the codebase is implicit, the Linux kernel has a cut-and-dried split into over 80 subsystems, each maintained by a group of maintainers following slightly different maintainership models [3, 25, 35]. Although the Linux kernel is often treated as an atomic project in mining software repository studies [18, 21], awareness of the existence of diverse, coexisting maintainership models supports both explanations of process-related phenomena and new research directions [31, 35]. For instance, a subsystem maintained by developers actively paid for by a given company might be at lower risk of turnover-induced knowledge loss, as some internal knowledge-sharing practices may not be captured solely by code metrics. Alternatively, one might wonder whether the number of contributors owning small parts of a given subsystem varies depending on whether it is maintained by a fluid group of core maintainers or by a single "benevolent dictator" [28].

We conduct a case study of a few notable subsystems of the Linux kernel to explore whether and how their varying maintainership models affect the distribution of two process-related metrics: the number of major and minor code owners. Balancing statistical insights with the authors' previous experience with varying Linux kernel development models, we verify the existence of non-negligible differences in how ownership evolves across Linux subsystems and discuss how contextual information about subsystems with different maintainership models helps explain how ownership correlates with local code metrics.

Our results go a step beyond just shedding light on the intricacy of how code ownership develops across the Linux kernel. The preliminary findings of our study serve as a testament to the importance of understanding how a codebase's maintenance and evolution are affected by the project's maintainership model and context. Among the implications of our findings, we speculate on a fruitful new direction for future work: augmenting repository mining with information on the contexts and policies in which different projects are developed and maintained to support contextually aware insights into software maintenance and evolution.

2 The Linux Kernel

The Linux kernel is frequently used as an ideal Free/Libre Open Source Software (FLOSS) test case in multiple software engineering studies due to its impressive size (over 27 million LoC) and rich history, spanning over 35 years of development [5, 22, 30]. Unlike most other FLOSS projects, however, the Linux kernel codebase is split into over 80 subsystems, each corresponding to a subset of files and folders, maintained in a separate git repository under the control of one (or more) maintainer(s).

While the existence of this split into subsystems is a fact, the limits of each subsystem are difficult to trace, as they rarely follow the boundaries of a single directory, including files from separate parts of the codebase [8]. Before being integrated into the official source code, a contribution is first reviewed by the maintainer(s) of the subsystem to which the change pertains, then reviewed again and tested by a hierarchy of maintainers who validate its inclusion using a chain-of-trust model [1, 24].

Each subsystem can adopt a different maintainership model: defining which maintainers are allowed to approve changes, how many maintainers should be active at any given time, and which communication and support tools will aid in maintaining the subsystem's code [9, 25]. Depending on the context of each subsystem, a varying number of companies might be interested in helping fund developers to preserve the Linux code, and the internal policy of these companies might, therefore, impact how the active maintainers of any given part of the kernel organize themselves and contribute to it.

As we previously acknowledged, the size and longevity of the Linux kernel codebase, by themselves, are a common argument for its relevance as a case study in mining software repositories. However, our choice to use it in our study stems from an even more defining characteristic of the Linux kernel. The variability enabled by the multiple maintainership models, balanced with the common background of a single project, makes the Linux kernel subsystems an ideal set of subprojects in which we seek to achieve this study's goal: **to explore how process metrics are affected by the policies, contexts, and interests that reign over different development teams**.

3 Study Design

For this case study, we examined the distribution of two consolidated process metrics: the numbers of major and minor owners for every file and directory in the Linux kernel codebase [4, 12, 33]. For every file and directory f in the Linux kernel, we define the number of major owners to be the smallest number of contributors who add up to 50%[2] of f's total contributions. More formally, given 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 number of major owners $Major(f)$ is defined as follows:

$$Major(f) = \min |S| \; s.t. \; C(S, f) \geq 0.5 \times C(D_f, f) \tag{1}$$

We follow previous work on developer ownership and expertise [10, 13, 26] and adopt the line-level blame of a given file or directory to evaluate a developer's contribution to it. That is, we consider Equation (1) with $c(d, f)$ being the number of lines attributed to developer d on the current version of file or module f, and $C(D_f, f)$ being its total size in lines. If f is a directory, we sum the contributions of d to every file recursively contained in it.

On the other hand, we define the number of minor owners of a file or directory f to be the number of developers who still have a contribution attributed to them in the git blame of f (or its subcontents), and who are not major owners, that is:

$$Minor(f) = |D_f| - Major(f) \tag{2}$$

[2]The 50% threshold is inspired by previous work. We leave for future work the merit of exploring varying thresholds for discerning major and minor code owners.

We chose to explore major and minor code owners in our case study because the relation between these metrics and fault proneness has been extensively studied in the past, but with significantly different results. While some studies have found strong evidence suggesting that weak code ownership is a common risk factor for faulty files and modules [4, 14, 33], others have suggested that such a connection between concentrated ownership and fault proneness does not exist, or is deeply nuanced [12, 26].

We hypothesize that the development model a module is in plays a pivotal role in whether strong ownership concentration induces fault proneness, and in how ownership affects codebase evolution and maintenance in general. Therefore, these ownership metrics serve our case study well. They provide a measurable dimension to the variability in the development model of the different subsystems, possibly validating the existence of concrete, objective differences in how these subsystems are evolved and maintained.

3.1 Study Objectives

Given our goal presented in Section 2 and our context on ownership metrics on the Linux kernel subsystems, we derive two research questions that guide our case study.

RQ1) *To what extent do the numbers of major and minor owners vary across subsystems?*

First, we would like to understand whether there is a fundamental difference in how the distribution of major and minor owners varies across the files and modules of different Linux subsystems.

RQ2) *How do the correlations between ownership metrics and local metrics vary across subsystems?*

Additionally, we explore whether local characteristics of files and modules within Linux subsystems interact differently with code ownership. We investigate the relationship between local metrics (node creation date, last modification, size, and number of commits) and the distribution of major and minor owners. Then, we examine how these relationships vary across subsystems.

3.2 Data Collection

Since git was developed as a side product of Linux kernel development, version control information from the first years of the codebase is not available in its official GitHub mirror (created in 2010). Therefore, we complement the data currently hosted on it with historical data from other git mirrors to include its complete development since September 1991. We analyze the code from the Linux kernel release 6.18, from November 2025.

While identifying the limits of the Linux subsystems, we refer to two previous works [3, 25] and the MAINTAINERS file, as advised by the official Linux developer community documentation [16]. We focus our case study on seven notable subsystems: **drm,** containing code used for interfacing with GPUs; **kernel,** also known as Core, contains the implementation of the core functions of the OS; **fss,** containing the source for the file systems supported by the Linux OS; **iio,** containing drivers for Industrial Input Output devices of multiple vendors; **i915,** subset of the drm subsystem, containing code for Intel integrated GPUs; **media,** containing code used for interfacing with media hardware; and **armSoC,** containing System on a Chip (SoC) code for ARM;

Table 1: Local metrics for the subsystems in our sample.

Subsystem	Files / Dirs	Creation Date	Last Modification	File Length (LoC)	No. Commits
drm	6653 / 548	Jul, 2019 ± 4.04 y.	May, 2023 ± 2.69 y.	1108.77 ± 6884.12	23.30 ± 63.61
kernel	2689 / 116	Jul, 2014 ± 6.86 y.	Oct, 2023 ± 2.36 y.	435.81 ± 979.55	50.77 ± 119.68
fss	2261 / 97	Aug, 2012 ± 7.16 y.	Mar, 2024 ± 2.22 y.	697.25 ± 1182.03	81.37 ± 178.28
iio	1361 / 87	Dec, 2018 ± 3.99 y.	Feb, 2024 ± 2.06 y.	331.62 ± 448.81	14.26 ± 22.53
i915	871 / 14	Jul, 2020 ± 2.64 y.	Mar, 2024 ± 1.67 y.	469.92 ± 857.80	54.44 ± 205.53
media	2972 / 201	Dec, 2016 ± 4.84 y.	Dec, 2022 ± 2.40 y.	505.50 ± 812.39	12.85 ± 20.87
armSoC	951 / 64	Nov, 2012 ± 4.57 y.	Sep, 2021 ± 2.29 y.	159.05 ± 253.92	24.09 ± 41.88
all	90572 / 6038	Jun, 2017 ± 5.99 y.	May, 2023 ± 2.67 y.	456.21 ± 2085.28	24.97 ± 105.50

For each file and module in the Linux kernel codebase, we compute the number of major and minor code owners using Equations (1) and (2), based on information collected with the git-blame and git-log commands. We identify developer aliases using Jaro-Winkler similarity between authors' names and email addresses, following the approach of Orrei *et al.* [23], since their ownership metric closely matches our definition of $Major(f)$. Following the Linux kernel's Developer Certificate of Origin, every contribution accepted into the kernel is signed by a human developer, so we do not have to exclude contributions from bots or CI/CD tools [2].

From a technical standpoint, our data extraction pipeline is written in both Bash and Python scripts. Since the core of the data collection is done using git commands, which are not parallelized by default, we use the GNU `Parallel` [32] utility to speed up data mining by exploiting the multi-core infrastructure at our disposal, mining data from multiple source files concurrently. In our statistical analyses, we use the `statsmodels` python library [29] for regressions and multicollinearity tests. All scripts used for data extraction and analysis are available in our replication package, which an interested reader is invited to explore in greater detail for further technical information.

Table 1 includes general statistics on the local variables collected for the different subsystems. In addition to the number of files and modules in each subsystem, we include the mean and standard deviation for each of our four local metrics for the files in that subsystem. While the number of subsystems may be small, they present notably distinct subprojects and account for 19.55% of the Linux kernel's LoC.

4 Exploring Ownership Concentration

We present the results of our two research questions and discuss them separately. Section 5 presents an overarching discussion with the implications of our case study as a whole.

4.1 RQ1) To what extent do the numbers of major and minor owners vary across subsystems?

As seen in Table 1, the average file length varies drastically across subsystems, with the armSoC subsystem's mean length close to one-third of the average for the complete Linux codebase, and the drm subsystem's mean length over 22 times that of the average Linux kernel file.

We understand that larger files and directories are more likely to have more contributors (both minor and major), since they more often include code fragments written for different purposes and responsibilities, and thus by different developers. Therefore, we decided to normalize the number of major and minor owners for each file and directory by dividing the number of developers by the file/directory's length. We refer to the normalized number of owners as the minor (respectively, major) owner *density*. We also see the need to note that density metrics could be inflated for exceptionally small files, an artifact we carefully keep in mind as we interpret the following results.

Figure 1 depicts the distribution of major (1.a) and minor (1.b) owner density across the files of the different subsystems.

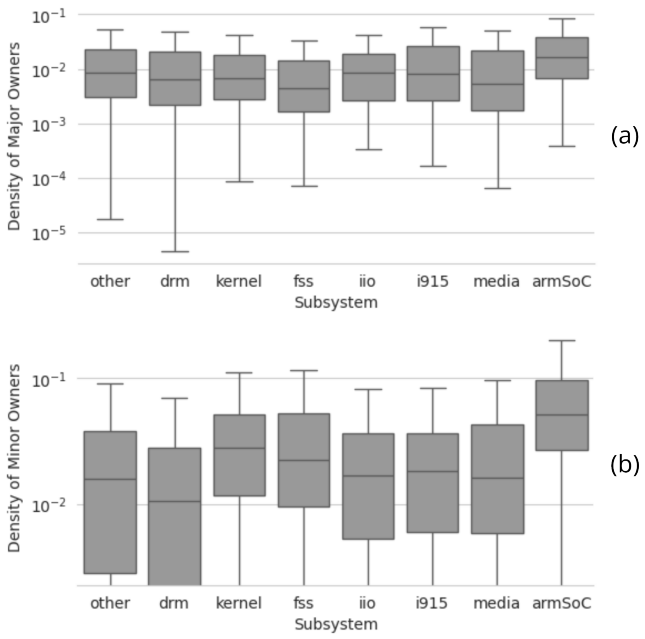

Figure 1: Density of major (a) and minor (b) owners on the files of the analyzed subsystems.

Two statistically significant Kruskal-Wallis tests confirm that the subsystems do not share a common distribution of major or minor owner density.

Further analyzing Dunn's test results adjusted using Benjamini Hochberg p-value correction reveals that most subsystems follow individually distinct distributions of major owners, except for two clusters of subsystems for which the null hypothesis of equal major owner density distribution is not rejected: one cluster with iio, kernel (core), i915, and the files outside the seven studied systems (other), and another cluster with the drm and media subsystems. The same analysis for minor owner density reveals two other clusters: one with kernel (core) and fss, and another with iio, i915, media, and the files from the remainder of the kernel.

Considering the density of major owners shown in Figure 1a, two subsystems draw attention: armSoC, which has a higher major owner density, and fss, which has the lowest. In fact, we note a small Cliff's Delta effect size between the distribution of major ownership density in the whole kernel compared against that of armSoC (0.254) and fss (0.205), with a medium effect size of 0.444 between them. Similarly, for minor ownership density, we observe non-negligible Cliff's Delta effect sizes between every pair of subsystems that are not identified in the same clusters determined by Dunn's test.

RQ1 — Discussion: Our results confirm that the maintainership models of the different subsystems have a statistically significant impact on the distribution of major and minor code owners. In the case of major ownership density, we see the media and drm subsystems clustered with stronger code ownership, both composed of drivers and low-level software that interface with GPUs and audio/video components.

Bigger still is the difference the armSoC subsystem has in major ownership density compared to all other subsystems, suggesting its unique "Hands-off" group maintainership model [25] favors collaboration among multiple major contributors to its files, despite them being smaller on average than those of the other subsystems (See Table 1). Regarding minor ownership density, we again note the prevalence of non-negligible effect sizes across most subsystems split across different clusters under Dunn's test. Notably, the average file size (as well as its dispersion measured by standard deviation) appears to strongly influence the clusters of subsystems with equivalent minor owner density distributions.

4.2 RQ2) How do the correlations between ownership metrics and local metrics vary across subsystems?

For the remainder of this section, we focus exclusively on Spearman correlations found to be statistically significant ($p < 0.05$) when adjusted using Bonferroni correction.[3]

When aggregating files across all kernel subsystems and considering the number of major owners per node, we observe two moderate correlations with folder size (0.469) and the number of commits to files (0.413). On the other hand, we find moderate and strong correlations between the number of minor contributors against all metrics studied: the creation date of files (−0.527) and folders (−0.544), the date of most recent modification on folders (0.620), length of files (0.596) and folders (0.812), and number of commits to a file (0.831).

[3]Although more conservative than the Benjamini-Hochberg correction used in RQ1, the Bonferroni correction is easier to implement and still yielded statistically significant results in the analyses conducted in RQ2.

The strong correlations observed with the number of commits and node length, in particular, support our previous decision to normalize the owner count by node size, but do little to characterize ownership concentration in general. Nevertheless, when comparing the magnitudes of these correlations across subsystems, we observe interesting changes.

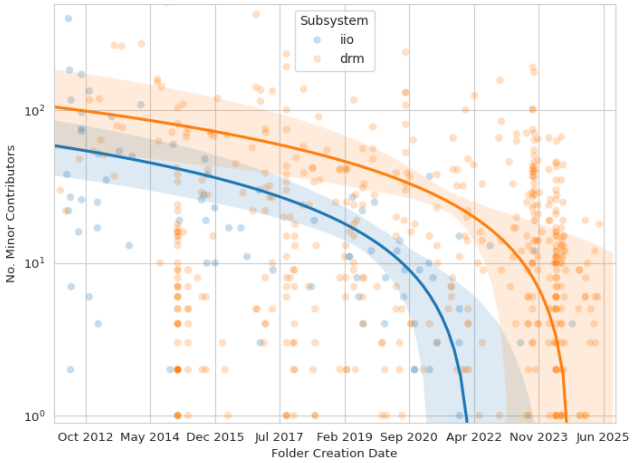

Figure 2: Number of minor contributors by subdirectory age in the iio and drm subsystems.

Figure 2 presents the number of minor owners and creation date of folders in the iio and drm subsystems. While iio has a Spearman's correlation coefficient of −0.625, the same two variables have a correlation coefficient of −0.241 in drm. Thus, new iio subdirectories present stronger concentration than newly created drm ones.

As seen in Figure 3, the correlation between the date of the last commit and minor contributor density varies notably between the media (−0.236) and kernel/core (−0.082) subsystems.

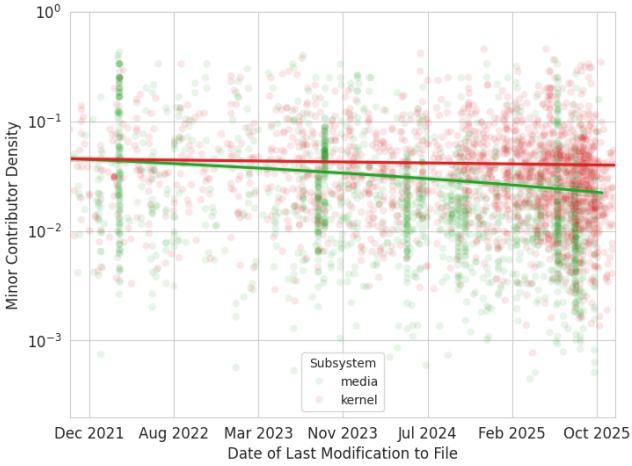

Figure 3: Density of minor contributors by time of last modification for files in the media and kernel subsystems.

Interpreting the plot visually, the red-colored cluster to the right of the plot shows that most files in the core subsystem are modified independently and frequently. Conversely, the patterns of green vertical stripes reflect how many files on the media subsystem are modified infrequently, and in batches. As a result of these batch updates, the weak negative correlation observed for the media subsystem appears to be an artifact, indicating no real connection between minor owner density and time since last modification.

RQ2 — Discussion: The two pairs of correlations with varying strength across the subsystems in our sample serve to show how the context and maintainership models of different subsystems affect how local metrics relate to ownership.

Our first example shows a stronger tendency for newly created folders to have fewer minor developers in iio than in drm. On the one hand, the drm subsystem is maintained by groups of highly active teams of professional developers collaborating on large drivers for consumer-grade GPUs, which favors contributor collaboration soon after the creation of new source code files. On the other hand, the iio subsystem is a classic example of a subsystem maintained predominantly by a single maintainer, and new contributions tend to be self-contained driver implementations for small industrial sensors and converters [25]. In this case, both the context and the maintainership model explain drastic changes in how code is co-authored across the two subsystems.

Our second example compared subsystems with files of similar average length but showed, for the media subsystem, an alleged tendency for files with more minor owners to go longer periods without receiving modifications. This tendency, however, is an artifact reflecting the drastically different purposes of the two subsystems within the larger context they fit into. Essentially, the files in the media subsystem are modified in batches and less frequently because they are families of drivers and audio/video protocol implementations. Meanwhile, the kernel core files are often dependencies of other features of the Linux kernel and are therefore modified more frequently. Being unaware of the distinction in the purposes of these two subsystems could lead a naive analyst to explore a spurious, misleading correlation.

5 General Discussion

The study of process metrics, including major and minor code ownership metrics, has been extensively studied in the past [4, 12, 33]. However, the interplay between the metrics and the context surrounding the development of different codebases remains severely underexplored in the large-scale scope adopted by most recent studies that mine software repositories. Our case study demonstrates how context-aware software repository mining can deepen quantitative and qualitative analyses.

Ultimately, the results of our case study demonstrate the practical impacts of the maintainership models of seven select Linux subsystems and how two ownership metrics are distributed across them. The results of RQ1 show that the context and maintainership models explain statistically significant differences in how ownership is concentrated across Linux subsystems, while the results of RQ2 show that the same contextual factors affect how ownership relates to local code metrics.

5.1 Implications and Future Work

The results of our case study urge for a new perspective for software repository mining studies: An approach that balances the qualitative insights derived from awareness of the organizational and financial context in which a codebase is developed, with the quantitatively backed statistical rigor of large-scale mining software repository (MSR) studies. While we hope our results, by themselves, serve as a wake-up call, we dedicate this subsection to discussing possible research directions that explore this new research perspective, following from noteworthy aspects of our results.

For instance, the comparison of the media and kernel (core) subsystems discussed following Figure 3 suggests that the participation of corporately-backed contributors can have a clearly perceptible impact on how a (sub)project evolves and on the correlations that might emerge in process-level metrics. The sparse clusters of change in the media subsystem reflect how that subproject was split across the turf of a select number of big players, while the widespread recency of changes in the core subsystem reflects its overall importance to most stakeholders involved in the development of the Linux kernel. We therefore hypothesize that the number of corporately backed FLOSS contributors, as well as the number of distinct backing entities, can be a defining factor in the maintenance of a large FLOSS project, potentially affecting the external validity of many currently practiced MSR studies.

Future work may explore the extent to which our hypothesis holds on different FLOSS projects, comparing them to freelance-only or closed-source software. Also towards the goal of probing our hypothesis's generalizability, a direct next step following our study would be to establish an effective, rigorous, and reproducible framework that formalizes the steps researchers and practitioners must adopt to conduct context-aware software repository mining. Such a framework could specify which context-related variables MSR studies should be aware of and thus pave the way for more intricate software repository mining pipelines that facilitate the inclusion of these variables in future studies.

Moreover, the considerable differences between the Linux subsystems in our initial sample invite future research on whether these subsystems (or projects) could be categorized by aspects of their context or maintainership models. Do subsystems that adopt different policies and tools for reviewing and accepting contributions often show significant changes in ownership (or other process metrics), as did ArmSoC and the filesystems subsystem in RQ1? Does clustering subprojects with similar target audiences help explain how collaboratively they are authored, as we observed in the first example of RQ2? Since Linux subsystems have a degree of autonomy in their maintainership models, could comparing FLOSS projects organized under different policies yield equally interesting results for process metrics?

Additionally, while we adopted code ownership in our preliminary case study, we find it reasonable to assume that other process metrics may also be significantly affected by organizational factors. The extent to which different process metrics are influenced by the project's context, community, and development model should be explored in future work, thereby corroborating the importance of context-aware repository mining studies.

We understand that most implications of our case study are of direct interest to software engineering researchers. Still, our results also have a direct effect on software engineering practitioners. Our results evidence that code ownership metrics are heavily dependent on the development model and context of any given project. Practitioners should be cautious when deciding which code ownership policy to adopt, as correlations observed in other systems may not apply to their context, given the different nature of the code they maintain or the organization the developers work in. More generally, practitioners should take a grain of salt when reinterpreting and repurposing the results of any study involving process-level metrics computed in projects with contexts dissimilar to theirs.

5.2 Limitations

Like any study, ours has limitations. Besides pointing out new avenues for future work, the example questions we just presented stress the limits of the external validity of our results. While the theoretical result that context and maintainership models influence ownership metrics is readily generalizable, the extent to which this occurs should be studied in future work using different process metrics and project samples.

We further acknowledge that our findings may be affected by threats to construct validity, as the heuristic we used to identify developer aliases and the metric we used to compute major and minor owners could be flawed. Similarly, confounding factors could affect the internal validity of our results (a Variance Inflation Factor analysis revealed collinearity between the dates of the first and most recent commits). Future work and replications will test whether our findings stand.

6 Related Work

While ownership metrics have been extensively studied in the past [4, 10, 33], we refer to four past studies notable for their focus on process metrics in Linux subsystems.[4] To the best of our knowledge, the first attempt to categorize the different maintainership models adopted on Linux kernel subsystems was the work of Pinheiro [25], which is especially relevant to ours. Their multivocal literature review also delves into blog posts and other grey literature from the Linux kernel community to identify subsystems with group maintainership models that contrast with the common figure of a single "benevolent dictator" maintaining each subsystem.

Notably, Avelino *et al.* [3] studied co-authorship networks in the Linux kernel, comparing the average contribution load and the context specificity of developers' work across six subsystems, finding different proportions of specialist and generalist contributors. Zhou *et al.* [35] also observed noticeable differences in code and contributor churn across Linux subsystems. Tan *et al.* [31] conduct an in-depth case study of the i915 subsystem following its successful adoption of a group maintainership model, and its positive impact on maintainer workload. Our work contributes to the state-of-the-art alongside these prior works by demonstrating how the context and maintainership model adopted by these subsystems explain the differences between them.

[4]As mentioned in Section 2, the definition of Linux subsystems is not explicit. Unlike us, most of the cited prior work adopts a different conception of what a subsystem accounts for, following the codebase's folder hierarchy.

7 Conclusion

We presented the results of a case study comparing differences in ownership concentration across subsystems of the Linux kernel. By considering the context and maintainership model adopted by seven subsystems, we explored the extent to which process metrics extracted from source code are affected by the policies, priorities, and organizational structures of the developers who authored the software. Beyond exploring ownership concentration in the Linux kernel, our study calls for future work in a fundamentally different research direction on mining development process metrics. Our results have direct implications for the research community, shedding light on a unique research direction that augments the well-established field of software repository mining by categorizing software projects and subprojects based on the context, domain, and managerial policies that govern how their developers maintain them.

Artifact Availability

Our replication package includes the scripts used to collect our ownership metrics in the Linux kernel codebase, our data, and the code used for our analyses. The package is available at: https://doi.org/10.6084/m9.figshare.32305320

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).

References

[1] [n. d.]. How the development process works: How patches get into the Kernel. https://www.kernel.org/doc/html/v6.17/process/2.Process.html#how-patches-get-into-the-kernel [Online; Last accessed on May. 10th, 2026].

[2] The kernel development community 2001. *Submitting patches: the essential guide to getting your code into the kernel.* The kernel development community. Retrieved May. 10th, 2026 from https://docs.kernel.org/process/submitting-patches.html#sign-your-work-the-developer-s-certificate-of-origin

[3] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2017. *Assessing Code Authorship: The Case of the Linux Kernel.* IFIP Advances in Information and Communication Technology, Vol. 496. Springer, 151–163. doi:10.1007/978-3-319-57735-7_15

[4] 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 SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE).* ACM, 4–14. doi:10.1145/2025113.2025119

[5] Ivan T. Bowman, Richard C. Holt, and Neil V. Brewster. 1999. Linux as a case study: its extracted software architecture. In *Proceedings of the International Conference on Software Engineering (ICSE).* ACM, New York, NY, USA, 555–563. doi:10.1145/302405.302691

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

[7] Melvin E Conway. 1968. How do committees invent. *Datamation* 14, 4 (1968), 28–31.

[8] Jonathan Corbet. [n. d.]. Finding real-world kernel subsystems. https://lwn.net/Articles/844539/ [Online; Last accessed on May. 10th, 2026].

[9] Jonathan Corbet. 2001. *Group maintainership models.* Retrieved May. 10th, 2026 from https://lwn.net/Articles/705228/

[10] Valerio Cosentino, Javier Luis Canovas 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).* IEEE, 499–503. doi:10.1109/SANER.2015.7081864

[11] Otávio Cury and Guilherme Avelino. 2025. The Impact of Generative AI on Code Expertise Models: An Exploratory Study. In *Proceedings of the 39th Brazilian Symposium on Software Engineering (SBES).* SBC, 706–712. doi:10.5753/sbes.2025.11074

[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)*. ACM, 1–9. doi:10.1145/2601248.2601283

[13] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. 2005. How developers drive software evolution. In *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE'05)*. 113–122. doi:10.1109/IWPSE.2005.21

[14] Michaela Greiler, Kim Herzig, and Jacek Czerwonka. 2015. Code ownership and software quality: a replication study. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR) (MSR '15)*. IEEE, Florence, Italy, 2–12. https://dl.acm.org/doi/10.5555/2820518.2820522

[15] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, New York, NY, USA, 92–101. doi:10.1145/2597073.2597074

[16] The kernel development community. [n. d.]. *How the development process works: Mailing lists*. Retrieved May. 10th, 2026 from https://docs.kernel.org/process/2.Process.html#mailing-lists

[17] Jiajun Li, Wenhua Yang, Minxue Pan, and Yu Zhou. 2025. Understanding Feature Request Practice on GitHub via a Large-Scale Empirical Study. In *Proceedings of the 40th International Conference on Automated Software Engineering (ASE)*. IEEE, 2783–2794. doi:10.1109/ASE63991.2025.00228

[18] Xiaozhu Meng, Barton P. Miller, William R. Williams, and Andrew R. Bernat. 2013. Mining Software Repositories for Accurate Authorship. In *Proceedings of the 29th International Conference on Software Maintenance (ICSM)*. IEEE, 250–259. doi:10.1109/ICSM.2013.36

[19] Mohit and Kuljit Kaur Chahal. 2026. Community engagement and the lifespan of open-source software projects. *Information and Software Technology* 189 (Jan. 2026), 107914. doi:10.1016/j.infsof.2025.107914

[20] Florent Moriconi, Thomas Durieux, Jean-Rémy Falleri, Raphaël Troncy, and Aurélien Francillon. 2025. GHALogs: Large-Scale Dataset of GitHub Actions Runs. In *Proceedings of the 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 669–673. doi:10.1109/MSR66628.2025.00104

[21] Mathieu Nassif and Martin P. Robillard. 2017. Revisiting Turnover-Induced Knowledge Loss in Software Projects. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 261–272. doi:10.1109/ICSME.2017.64

[22] Gianlorenzo Occhipinti, Csaba Nagy, Roberto Minelli, and Michele Lanza. 2023. SYN: Ultra-Scale Software Evolution Comprehension. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. IEEE, 69–73. doi:10.1109/ICPC58990.2023.00020

[23] Vincenzo Orrei, Marco Raglianti, Csaba Nagy, and Michele Lanza. 2023. 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, 2062–2066. doi:10.1145/3611643.3613085

[24] Rafael Passos, Arthur Pilone, David Tadokoro, and Paulo Meirelles. 2025. Streamlining Analyses on the Linux Kernel with DUKS. In *Proceedings of the Working Conference on Software Visualization (VISSOFT)*. 125–128. doi:10.1109/VISSOFT67405.2025.00025

[25] Eduardo Pinheiro and Paulo Meirelles. 2024. Understanding Group Maintainership Model in the Linux Kernel Development. In *Proceedings of the Workshop on Software Visualization, Maintenance and Evolution (VEM)* (Curitiba/PR). SBC, Porto Alegre, RS, Brazil, 113–124.

[26] Foyzur Rahman and Premkumar Devanbu. 2011. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. ACM, 491–500. doi:10.1145/1985793.1985860

[27] Foyzur Rahman and Premkumar Devanbu. 2013. How, and why, process metrics are better. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 432–441. https://dl.acm.org/doi/10.5555/2486788.2486846

[28] Eric S. Raymond. 1998. Homesteading the noosphere. *First Monday* 3, 10 (1998). https://firstmonday.org/ojs/index.php/fm/article/view/621

[29] Skipper Seabold and Josef Perktold. 2010. statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference (scipy)*.

[30] Roman Suvorov, Meiyappan Nagappan, Ahmed E. Hassan, Ying Zou, and Bram Adams. 2012. An empirical study of build system migrations in practice: Case studies on KDE and the Linux kernel. In *Proceedings of the International Conference on Software Maintenance (ICSM)*. IEEE, 160–169. doi:10.1109/ICSM.2012.6405267

[31] Xin Tan, Minghui Zhou, and Brian Fitzgerald. 2020. Scaling open source communities: an empirical study of the Linux kernel. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*. ACM, 1222–1234. doi:10.1145/3377811.3380920

[32] Ole Tange. 2023. *GNU Parallel 20231122 ('Grindavík')*. doi:10.5281/zenodo.10199085

[33] Patanamon Thongtanunam and Chakkrit Tantithamthavorn. 2024. Code Ownership: The Principles, Differences, and Their Associations with Software Quality. In *Proceedings of the 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 379–390. doi:10.1109/ISSRE62328.2024.00044

[34] Xiao Yu, Lei Liu, Xing Hu, Jin Liu, and Xin Xia. 2025. Where Is Self-admitted Code Generated by Large Language Models on GitHub?. In *Proceedings of the 32nd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 407–418. doi:10.1109/APSEC66846.2025.00047

[35] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. 2017. On the scalability of Linux kernel maintainers' work. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, Paderborn Germany, 27–37. doi:10.1145/3106237.3106287