The Dynamics of Knowledge Concentration in Long-lived Collaborative Software Codebases

Arthur Pilone*
arthurpilone@ime.usp.br
Institute of Mathematics, Statistics, and Computer Science
University of São Paulo
São Paulo, Brazil

Abstract

The organizational factors that govern a development team, such as which parts of the code each developer works on or how code changes are reviewed, are pivotal to the survival of large software codebases. Balancing such organizational factors is an even greater challenge for Free/Libre and Open Source Software (FLOSS) projects, which must sustain hierarchies of maintainers while preventing any of them from becoming overloaded with too much responsibility or concentrating too much knowledge of the codebase. This PhD project aims to identify the effects of different development models on the evolution of knowledge concentration in large, long-lived FLOSS projects. To that end, we envision three phases for the project, each with its expected contribution: (A) A new metric to quantify knowledge concentration considering commit authorship and reviewing duties; (B) A tool to visualize the evolution of knowledge concentration in software codebases; and (C) A taxonomy of patterns of development models for mitigating knowledge-concentration-induced bottlenecks in collaborative codebases.

CCS Concepts

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

Keywords

Development Models, Knowledge Concentration, Software Visualization, Software Maintenance.

ACM Reference Format:
Arthur Pilone. 2026. The Dynamics of Knowledge Concentration in Long-lived Collaborative Software Codebases. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3774748.3787663

1 Background

Decades of software engineering research and practice crystallize the idea that the managerial and organizational aspects of a software project play a vital role in its success (or failure) [3, 4]. We can refer to the set of development practices and values that culminate from a project's organizational factors as its development model.[1]

One of the most crucial decisions in a project's development model is determining who should write the code and who should review it. In the case of collaborative and Free/Libre and Open Source Software (FLOSS) projects, this commonly manifests as a subgroup of dedicated contributors, often referred to as maintainers, who are responsible for ensuring the quality of the accepted contributions. Such is the case for the Linux kernel, where only select maintainers are authorized to approve incoming contributions, which they sign with the `Signed-off-by` tag [1, 15].

Failing to balance, on the one hand, the quality control of having only a few experienced maintainers responsible for the review process against, on the other hand, the dangers of having overloaded maintainers is a recurring threat that commonly undermines a codebase's future [11]. Returning yet again to the Linux kernel as an example, grey literature on the maintainers' perspectives suggests possible scalability issues as the number of contributors rises faster than the number of maintainers [21, 22]. When considering whether to accept a contribution, maintainers must weigh the burden of also maintaining the newly introduced code in the future, avoiding a contribution if they understand it might overload them with future responsibilities [6].

In an effort to mitigate the risks of having maintainers overloaded with too much responsibility due to their extensive knowledge of the system, past authors have proposed metrics to quantify knowledge concentration and studied the risks associated with developer turnover in FLOSS [13, 18]. However, most knowledge concentration metrics consider only a developer's code changes and do not take into account their roles in reviewing code [2, 9].

Few studies explored adding code reviewing activities to computing knowledge concentration metrics obtaining positive results [11, 19]. Nonetheless, to the best of our knowledge, no past work has proposed quantifying maintainer knowledge and overload with a specific metric that fuses both the code changes FLOSS maintainers author and the other 'auxiliary' duties that take up most of their time. Consequently, the state-of-the-art lags behind on how knowledge concentrates in the majority of FLOSS projects, which are highly collaborative and review-dependent by nature. In particular, it is still unclear how different development models contribute to or avoid overloading maintainers with excessive responsibility.

*PhD student, in year 2 out of expected 5, advised by Paulo Meirelles and Fabio Kon

[1] This may include its maintainership model [17] or governance model [14].

ICSE-Companion '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2296-7/2026/04
https://doi.org/10.1145/3774748.3787663

2 Objectives and Research Questions

We define three research questions (RQs) to trace out the main objectives of this PhD project. Following the research gap identified in Section 1 and as a culmination of our own interest in understanding how development models contribute to or mitigate knowledge concentration, we pose our most fundamental research question: **(RQ$_a$) What are the impacts of different development models on the evolution of knowledge concentration?**

Given the complex nature of assessing the evolution of knowledge concentration, we set forth a second goal that will inform our methodology for answering RQ$_a$. Exploring the potential of software visualization for program maintenance and comprehension, we ask: **(RQ$_b$) What insights can a carefully crafted visualization elicit on the evolution of knowledge concentration on a long-lived software project?**

Since a visualization-powered analysis depends on high-quality data to be analyzed in the first place, our final research question will guide us in designing a new metric that will be the focus of our visualization. Corresponding to another research gap noted in Section 1, we will explore how a knowledge metric composed of an author's commit and review activities compares to existing knowledge and authorship concentration metrics. We therefore ask: **(RQ$_c$) Does accounting for review activity improve the accuracy of knowledge concentration metrics for large FLOSS projects?**

3 Methodology

This study employs a mixed-methods approach [7], varying across different research phases. We began with an initial exploratory phase to gain a deep understanding of the FLOSS ecosystem. We contributed to the Linux kernel and collected data from other contributions made by our research group. The insights from this foundational exploratory work informed the design of the presented research questions and guided us to our current experimental results.

Moving forward, and reflecting on the threefold nature of our research objectives (and, consequently, our RQs), we organize this PhD project into three different work packages (WPs). We will move to develop and validate a tailored knowledge concentration metric by combining case studies with semi-structured interviews [7]. Following this, our research will focus on developing and validating a visualization tool to analyze the evolution of this metric. Finally, we will utilize this validated approach to conduct large-scale comparative studies to investigate how different development models impact the dynamics of knowledge concentration.

We will first tackle RQ$_c$ with WP$_c$ – an initial round of case studies with FLOSS projects and semi-structured interviews with their maintainers will guide the design of a new knowledge concentration metric. Interactions with practicing maintainers will be essential for tuning the parameters of the new metric, and the open-ended questions could capture additional insights into how they perceive the burden of reviewing compared to authoring code. We shall conduct an evaluation comparing our new metric with related knowledge concentration metrics and previous studies, such as those of Ferreira *et al.* [8] and Hajari *et al.* [11].

WP$_b$ will center around RQ$_b$ and our visualization framework. Our new visual metaphor will balance the strengths of different consolidated visual metaphors, such as the broad system overview easily parsed from the Code City metaphor [16, 23], or the color-coded approach to portraying code authorship from the Ownership Map metaphor [10, 12]. Usability tests [5, 20] with another set of FLOSS maintainers will validate the implementation of the new visualization framework and test whether it is capable of supporting analyses on the dynamics of knowledge concentration.

Exploring the results from both WP$_c$ and WP$_b$, WP$_a$ will base our investigation on RQ$_a$. First, a warm-up study with three different subsystems from the Linux kernel adopting different development models [17] will validate whether our approach can be used to identify differences in the evolution of knowledge when comparing different codebases. Then, a final round of case studies with other FLOSS projects (*i.e.,* Git, Open SSL, GCC, Chromium) supported by our visualization framework will serve as the basis for comparisons between development models and, ultimately, guide us in identifying their impacts on the evolution of knowledge concentration.

4 Current Work

As an initial exploratory cycle on the nature of the Linux development model, we have examined whether quantitative data support the claims of unsustainability in the Linux development model mentioned in Section 1. In the first cycle, we engaged in the Linux development model, contributing to the `linux-iio` (Industrial Input/Output) subsystem.[2] This experience was essential for understanding aspects of the Linux contribution model that outsiders rarely note. From the idealization of the contribution and the setup of the development environment, to the conventions expected from commit messages and the process of sending contributions by email, this participant observation experiment made us aware of what it takes to contribute to a large, long-lived FLOSS project.

After contributing to Linux, we developed a visual framework for assessing various metrics in the project's code review process. This framework culminates on `DUKS`, the *Dashboard for Unified Kernel Statistics* [15]. It serves both as an ongoing work on software visualization and as part of the main objectives of this research project, offering an approach to compare how contributors' and reviewers' activities change on Linux subsystems with different development models.

As a follow-up to our experience with the `DUKS` analytical dashboard, our focus has shifted to mining new development metrics from mailing lists, in an approach tailored specifically to the Linux development model. Leveraging both the insights gained from contributing to the kernel and our interest in using quantitative data to assess the health of the Linux subsystems development models, we composed an extensive dataset of over 20 million emails collected from the Linux mailing lists. The data collected includes vital information on how frequently auxiliary contributors perform code reviews, as identified by the `Reviewed-by` and `Acked-by` tags [1, 15], and will serve as a basis for our first attempts at defining our compound knowledge concentration metric.[3]

[2] https://lore.kernel.org/linux-iio/20250421145534.91146-1-arthurpilone@usp.br/t/#u
[3] A preview of the dataset is available at: https://tinyurl.com/zenodo-lkml5ws

5 Expected Contributions

We expect to identify the practices and values adopted by diverse communities that influence how knowledge concentration evolves on their codebases. The answer to RQ_a could be used to support guidelines on the patterns for controlling knowledge concentration on long-lived collaborative codebases.

Following the reasoning behind RQ_b, we will develop a new tool that explores a visual metaphor to convey changes in knowledge concentration. The new metaphor will also be effective at conveying how the knowledge concentration varies with time, reflecting not only the increase or decrease in concentration, but also the speed at which these changes occur.

Finally, as part of our investigation on RQ_c, we will design, implement, and evaluate a new metric tailored for quantifying knowledge concentration. After answering RQ_c, we will have empirically validated this metric as adequate for quantifying knowledge concentration in FLOSS projects.

The contributions related to RQ_c and RQ_b are to be validated in this research project itself. The proposed knowledge concentration metric will be evaluated in WP_c, and the new visualization will be evaluated in WP_b. Moreover, ensuring the reproducibility of our studies in accordance with open science principles will enable interested researchers to easily validate and replicate our evaluations for RQ_c and RQ_b. This is facilitated by the fact that the object of our study is FLOSS, meaning the data is readily available. We will also release our datasets openly, the developed software as FLOSS, and provide replication packages.

As a consequence of its exploratory and analytical nature, other works will be able to validate the contributions resulting from RQa by examining our replication packages or by reproducing and expanding our analysis to assess new or previously unexplored FLOSS projects with different development models.

Acknowledgments

This work is financed, in part, by CAPES (Finance Code 001), and the São Paulo State Data Analysis System Foundation - SEADE (FAPESP Proc. 2023/18026-8), Brazil.

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 Nov. 10th, 2025].

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

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

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

[5] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. doi:10.2307/249008

[6] Sam Dean. 2020. The maintainer's paradox: Balancing project and community. linuxfoundation.org/blog/the-maintainers-paradox-balancing-project-and-community/.

[7] W Alex Edmonds and Thomas D Kennedy. 2016. *An applied guide to research designs: Quantitative, qualitative, and mixed methods.* Sage Publications.

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

[9] Thomas Fritz, Jingwen Ou, Gail C. Murphy, and Emerson Murphy-Hill. 2010. A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, Vol. 1. IEEE, 385–394. doi:10.1145/1806799.1806856

[10] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. 2005. How developers drive software evolution. In *Proceedings of The International Workshop on Principles of Software Evolution (IWPSE)*. IEEE, 113–122. doi:10.1109/IWPSE.2005.21

[11] Fahimeh Hajari, Samaneh Malmir, Ehsan Mirsaeedi, and Peter C. Rigby. 2024. Factoring Expertise, Workload, and Turnover Into Code Review Recommendation. *Transactions on Software Engineering* 50, 4 (April 2024), 884–899. doi:10.1109/TSE.2024.3366753

[12] Lile Palma Hattori, Michele Lanza, and Romain Robbes. 2012. Refining code ownership with synchronous changes. *Empirical Software Engineering* 17, 4 (Aug. 2012), 467–499. doi:10.1007/s10664-010-9145-5

[13] 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

[14] Pedro Oliveira, Doris Amoakohene, Toby Hocking, Marco Gerosa, and Igor Steinmacher. 2025. Governance Matters: Lessons From Restructuring the Data.Table OSS Project. In *Proceedings of the 41st International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 1–12. doi:10.1109/ICSME64153.2025.00067

[15] Rafael Passos, Arthur Pilone, David Tadokoro, and Paulo Meirelles. 2025. Streamlining Analyses on the Linux Kernel with DUKS. In *Proceedings of the 13th Working Conference on Software Visualization (VISSOFT)*. IEEE, 125–128. doi:10.1109/VISSOFT67405.2025.00025

[16] Federico Pfahler, Roberto Minelli, Csaba Nagy, and Michele Lanza. 2020. Visualizing Evolving Software Cities. In *Proceedings of the 8th Working Conference on Software Visualization (VISSOFT)*. 22–26. doi:10.1109/VISSOFT51673.2020.00007

[17] Eduardo Pinheiro and Paulo Meirelles. 2024. Understanding Group Maintainership Model in the Linux Kernel Development. In *Proceedings of the 12nd Workshop on Software Visualization, Maintenance and Evolution (VEM)* (Curitiba/PR). SBC, Porto Alegre, RS, Brazil, 113–124.

[18] Peter C. Rigby, Yue Cai Zhu, Samuel M. Donadelli, and Audris Mockus. 2016. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM, Austin Texas, 1006–1016. doi:10.1145/2884781.2884851

[19] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2016. 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, 1039–1050. doi:10.1145/2884781.2884852

[20] Mark Turner, Barbara Kitchenham, Pearl Brereton, Stuart Charters, and David Budgen. 2010. Does the technology acceptance model predict actual use? A systematic literature review. *Information and Software Technology* 52, 5 (May 2010), 463–479. doi:10.1016/j.infsof.2009.11.005

[21] Daniel Vetter. 2017. Maintainers Don't Scale. blog.ffwll.ch/2017/01/maintainers-dont-scale.html [Online; Last accessed on Nov. 5th, 2025].

[22] Melissa Shihfan Ribeiro Wen. 2021. *What happens when the bazaar grows: a comprehensive study on the contemporary Linux kernel development model.* Ph. D. Dissertation. University of São Paulo.

[23] Richard Wettel and Michele Lanza. 2007. Visualizing Software Systems as Cities. In *Proceedings of the 4th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE, Banff, AB, Canada, 92–99. doi:10.1109/VISSOF.2007.4290706