Preprint: Streamlining Analyses on the Linux Kernel with DUKS

Rafael Passos, Arthur Pilone, David Tadokoro, Paulo Meirelles
Free Software Competence Center
Institute of Mathematics and Statistics
University of São Paulo, Brazil
{*rcpassos,arthurpilone,davidbtadokoro*}*@usp.br,paulormm@ime.usp.br*

***Abstract*—With its remarkably extensive code base, uniquely long lifespan, and undeniable importance to modern society, the Linux kernel is trivially hard to maintain. However, its decentralized development spread over many git trees and mailing lists makes empirically assessing the health of its maintainership model nothing short of a challenge. Off-the-shelf data analysis tools fail to capture crucial nuances exclusive to the kernel development model, such as the current authors who take part in every patch submitted, or how the commit flow between trees changes as new release candidates are created for every merge and stabilization window. We propose the `Dashboard for Unified Kernel Statistics` (`DUKS`), an innovative framework that supports multiple visualizations and data analyses previously unsupported for the Linux kernel. Using the Linux kernel mainline as an example, we demonstrate how `DUKS` could provide valuable insights for understanding the health of the kernel maintainership model. By coupling information from the kernel git trees collected from the `Software Heritage` repository alongside authorship information shared in mailing lists, we envision `DUKS` as a cornerstone open-access utility to support analyses on the Linux kernel evolution and maintenance. Link to the `DUKS` demo video [1].**

***Index Terms*—Linux kernel, Software Heritage, Visual Analysis, Free Software development model, Free Software maintainers**

I. Introduction

The Linux kernel is a monumental achievement in software engineering and is often regarded as the backbone of modern technological infrastructure. However, due to its long development history, the kernel workflow still incorporates original practices that reflect collaborative software development methods from over two decades ago [2].

The patch is at the core of the Linux kernel contribution model: a textual document representing the differences between two source code versions. A patch is essentially a *git commit* formatted to be emailed by its author. As a result, mailing lists, along with the associated discussions where these patches are shared, serve as the main communication channel between contributors and maintainers of the Linux kernel. These lists focus on development activities, bug reports, and technical discussions.

Contributors submit patches for review by maintainers, who provide feedback in an iterative *patch-reviewing process*. Through this process, maintainers ensure the quality of contributions, ultimately accepting them into the kernel codebase [3]. Once accepted, a set of patches *(patchset)* moves from the repository of the relevant subsystem to eventual integration into the mainline: the repository that centralizes the whole Linux kernel source code.

Due to its importance, the Linux kernel keeps growing in size, complexity, and number of contributors, which serves as a sign of constant evolution and maintenance of the project. Its complexity increases as many contributors continuously add new features, drivers, and components to the Linux kernel.

Academia and the community have raised worries about the sustainability of its maintainership model. Investigating the scalability of the kernel community and workflow, previous authors have observed empirical evidence indicating its current development model might be unsustainable [4]–[13].

Although the number of maintainers keeps growing steadily, as listed in the `MAINTAINERS` file, the number of mean contributions for each maintainer varies at a different pace [13]. Consequently, parts of the code can go long periods without new contributions [5]. Estimating the number of active maintainers poses a challenge in itself. Using exclusively the `MAINTAINERS` file can lead to false positives (maintainers listed but no longer active in the project) and false negatives (maintainers active but not yet listed in the file). More precise measurements require crosslinking data from multiple sources (*e.g.,*, mailing lists, *git history*), which becomes unfeasible without tools developed specifically for this task.

Conventional software mining techniques rarely factor in specificities as nuanced as which contributors took part in the discussion that led to a patch being rejected, or how every developer involved contributed by reviewing, testing, or acknowledging an accepted patch. Moreover, particularities of the Linux kernel itself, such as the split of every development window into a merge and stabilization phase [14], could bring additional information for comprehending and evaluating the periodic flow of contributions natural to the Linux kernel.

In this paper, we propose `DUKS` – **Dashboard for Unified Kernel Statistics**: with a novel approach for collecting, aggregating, and visualizing development metrics tailored to the Linux kernel maintainership model. We aim to integrate information from the patch flow in kernel mailing lists with the graph dataset provided by the largest publicly available source code archive, the `Software Heritage` project [15]. We

Preprint: paper accepted to IEEE VISSOFT 2025, Visualization Challenge Track.

also present a visualization enabled by our proof-of-concept implementation of DUKS, which illustrates the potential of the approach to support analyses on the health of the Linux kernel maintainership model.

II. Data Processing and Collection

Leveraging the fact that Linux kernel development is spread across public mailing lists and Git repositories, our approach aims to explore the openly accessible data on kernel development to support reproducible, empirically based analyses on the sustainability of its development model.

The first step is to collect data from the kernel mailing lists, where, for each subsystem, contributors send their patches, discuss with other contributors, and receive reviews from maintainers. We can collect this data from repositories like the *Kernel Lore Archives* and consolidate it into a structured format. A specialized data schema is necessary to store the patch contents and metadata, as well as information extracted from the email body and subject[1], such as patchset version or number of patches in a patchset, etc.. The designed structure will be stored in an open analytical format, such as Apache Parquet or ORC (Optimized Row Columnar), which allows for partitioning data by date for temporal analysis.

The second action is to use the source code archived by the Software Heritage initiative. By structuring the data related to every code revision in a graph, the Software Heritage repository supports mapping the evolution of multiple subsystems hosted on different trees. The Software Heritage is a public, deduplicated Merkle tree structure dataset, where all identical entities (revisions, releases, file names, file contents etc.) point to the same node even when found in different repositories [16]. This deduplication enables us to use these nodes as stable references for navigating across the various Git trees that make up the Linux kernel development history.

Our strategy is based on integrating the data extracted from the mailing lists into the VCS (*Version Control System*) dataset. Combining these overcomes the limitations of analyzing exclusively the Git history of these repositories, allowing the exploration of patch authorship, reviews, and how mailing list discussions relate to commits on different Git repositories.

For our **proof-of-concept** presented in this work, we he selected the compressed Software Heritage graph export [2]. Following storage limitations, we use the 1.5TiB subgraph *"History and hosting" Compressed graph* [17] shared in WebGraph [18] format. The subgraph contains revisions (*commits*) and origins (repositories), but no source code.

Using the origin node (repository name), we search for the latest snapshot available in the graph to retrieve releases *(git tag)* and the latest revision available. Starting on the most recent commit of the dataset, we retrieve all revisions of the kernel tree using a breadth-first search. We create a tabular dataset containing every commit message date, hash, and attributions (signatures). Given that we intend to cross the data from the commits with mailing lists and the MAINTAINERS file, we need to revert the pseudonimization process done by the Software Heritage project and obtain the email address for the contributors of every commit. To that end, we complement the data of our dataset with authorship information collected from the original git repository of every tree in the dataset.

To obtain a list of official maintainers, we search through the entries of the MAINTAINERS at the repository root in every file revision. We join this data into our revisions dataset using DuckDB[3]. Next, using the Polars[4] library, we aggregate commits based on their commit date to form a time series with metrics to be explored and studied. We parse the attributions of every commit message and compute intersections with MAINTAINERS for all points in time.

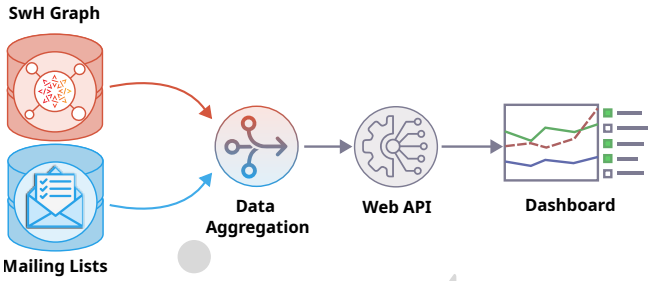

Fig. 1. High-Level Component Overview

Figure 1 illustrates an overview of the application components. We use high-level tools provided by Software Heritage, based on their lower-level library swh.graph[5] implemented in Rust. Our next increment to the implemented proof-of-concept is to develop a compatibility layer to access the mailing list entries and cross them with multiple Git trees from the original Software Heritage graph. The metrics are pre-calculated before being served to be presented by the proposed *Dashboard*.

III. A Dashboard for Unified Kernel Statistics

In the first version of DUKS, we present a time series visualization focused on maintainers' activity and workload. All data used in our analysis is sourced from the Software Heritage archive.

The strategy described in Section II provides a timeline of all observed contributions in the repository. However, the commit date of an accepted patch represents only the final step in the contribution process. According to Jiang *et al.* [19], most patches take three to six months to be integrated. Although this average may have changed since its publication, and varies depending on the complexity of each contribution, it highlights how much of the contributor's effort occurs before the recorded commit date. Therefore, when analyzing the involvement of a contributor, each accepted commit reflects

[1]Some relevant tags are presented in the subject message

[2]18/05/2025 export: docs.softwareheritage.org/devel/swh-export/index.html

[3]DuckDB Database Management System: duckdb.org

[4]Polars DataFrames: pola.rs

[5]All available at docs.softwareheritage.org/devel/apidoc/swh.graph.html

Preprint: paper accepted to IEEE VISSOFT 2025, Visualization Challenge Track.

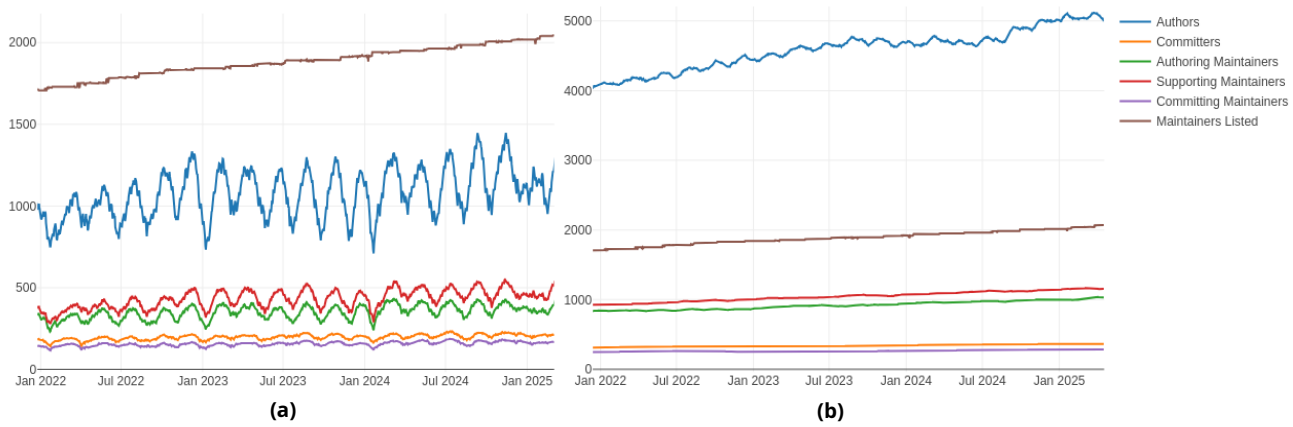

Fig. 2. Activity of Maintainers and Contributors in different roles

the culmination of potentially months of work and discussion. To account for this, we apply a rolling count to several metrics, capturing contributors who may have been active in the days, weeks, or months leading up to each commit.

In summary, the first version of DUKS, illustrated in Figure 2, is sufficient to explore and analyze some questions. The user interface is built as an HTML page featuring line charts generated with the Plotly library, selection boxes for data filtering, and VueJS for managing the interface state. All additional calculations are performed on the backend using `Polars`. Filtering commits by date intervals is straightforward and can be done by selecting a graph section with the cursor.

For example, one of the initial questions that motivated our work was: **How many active maintainers are there in the Linux kernel?** As mentioned earlier, the `MAINTAINERS` file lists those responsible for each *driver* and subsystem in the kernel. In Figure 2, we analyze this question in detail.

We use one-month or one-year windows for all time series, except for the *Entries in the `MAINTAINERS` file*, representing a fixed value at a given point in time. The graphs in Figure 2 displays, in decreasing order of magnitude: *Maintainers Listed* (number of maintainers registered in the `MAINTAINERS` file); *Authors* (number of commit authors); *Supporting Maintainers* (number of maintainers acting in roles other than *Author* or *Committer*); *Authoring Maintainers* (number of authors also listed as maintainers); *Committers* (number of individuals who committed changes); and *Committing Maintainers* (number of maintainers who made commits).

The graph (Figure 2.a) supports our assumption regarding the disparity between listed and active maintainers. This difference remains considerable even considering one-year rolling windows (Figure 2.b). While this disparity is not necessarily a sign of concern, given that many areas of the Linux kernel are stable and require little to no maintenance, the persistent gap between the *Committers* and *Committing Maintainers* series (the last two) indicates that not all individuals who commit code are formally listed in the `MAINTAINERS` file.

We have also investigated the release/stabilization cycle

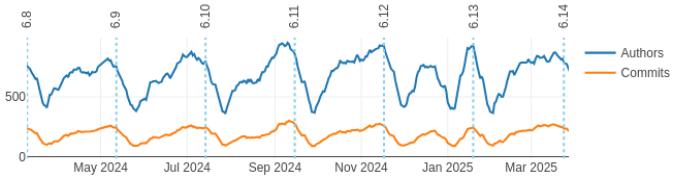

Fig. 3. Effects of the Release/Stabilization Cycle on Contributions

contributions, inspired by Rahman *et al.* [20]. In Figure 3, we show the average of commits and unique authors in a two-week window. As explored by the previous author, the impact of the date-centric management method is visible.

IV. Discussion and Final Remarks

The Linux kernel development workflow is predominantly based on discussions and reviews on its mailing lists. We envision DUKS, an innovative repository analysis tool tailored to the Linux kernel. It incorporates data from its mailing lists into the graph model used to represent code repositories in the `Software Heritage` dataset, enabling analyses previously not feasible. Navigating the nodes of this enriched graph could make it possible to trace the path of a given patch across Git repositories of multiple subsystems and trees, as submission messages and pull requests captured from the mailing lists can help reconstruct the patch flow within the kernel repository hierarchy.

No single commit message can fully reflect a contributor's time and effort in a patch. Mailing lists can help close this gap by preserving data that makes it possible to identify multiple versions of the same contribution, map technical discussions that shaped specific design decisions, and understand which community members contributed throughout the patch development. By integrating these complementary sources of information and accounting for the specific characteristics of the Linux kernel, our approach aims to facilitate the understanding of its contribution flow and to support analyses on the sustainability of the maintainership model adopted by one of the most critical software systems in modern computing.

Preprint: paper accepted to IEEE VISSOFT 2025, Visualization Challenge Track.

Recent studies explore the scientific value of the data extracted from the Linux kernel mailing lists. For instance, Schneider *et al.* [21] analyze differences in how leaders communicate in the kernel mailing lists; Ahmed *et al.* [22] investigate the utility of the feedback given in the mailing lists; and Hatta *et al.* [23] examine the role of mailing lists for Free Software development, focusing on the Debian project.

Moreover, GrimoireLab is a toolset for retrieving, enriching, and visualizing data about software development [24]. It is a powerful general-purpose tool, but not tailored to the Linux kernel. While it could be adapted to address some of the specific limitations we encountered, we chose to pursue a different approach. Although some of GrimoireLab purposes overlap with ours, our work accounts for details specific to the Linux kernel. For instance, to help identify contributors, we leveraged the `MAINTAINERS` file (unique to the Linux kernel), as well as attribution signatures, also known as *git trailers*. However, the GrimoireLab collector tool, called Perceval [25], discards all non-standard trailers[6].

Basing our work on the `Software Heritage` removes the need to index repositories ourselves. For our proof-of-concept, we created our index including **only** the repositories relevant to Linux kernel development. This approach eliminates the need for high-performance machines to handle the complete graph provided by the project. Additionally, we use a self-hosted stack configured via `swh-docker`[7]. With this setup, for example, we queue indexing tasks for the repositories listed in the `MAINTAINERS` file, which correspond to the project subsystems, drivers, and tools trees. While this graph view is not optimal for analyzing time series metrics, it enables different exploration strategies, algorithms, and a new class of graph-based visualizations. Nevertheless, it is still possible to derive time series structures from the graph, as demonstrated in our proof-of-concept.

V. Source code and Reproduction Package

The `DUKS` source code, licensed under GPL-3.0, along with the data required to reproduce our analyses and visualizations, is available at github.com/linux-duks/DUKS-2025-replication-pkg. We will continue to develop this project in its central repository at github.com/linux-duks/DUKS.

Acknowledgments

This study was financed, in part, by CAPES (Finance Code 001), the University of São Paulo - USP (Proc. 22.1.9345.1.2), the São Paulo Research Foundation - FAPESP (Proc. 19/26702-8) and the São Paulo State Data Analysis System Foundation - SEADE (Proc. 2023/18026-8), Brazil.

References

[1] "DUKS Demo Video," Youtube Link: youtu.be/2RvUgzdr1fo. [Online]. Available: https://archive.softwareheritage.org/browse/content/sha1_git:ecc718168d6fbf9d1baec830839db43c65979980/?origin_url=https://github.com/linux-duks/DUKS-2025-replication-pkg&path=/demo.x265.mp4

[2] J. Corbet, "Why kernel development still uses email," 10 2016. [Online]. Available: https://lwn.net/Articles/702177/

[3] N. Palix, S. Saha, G. Thomas, C. Calvès, J. Lawall, and G. Muller, "Faults in linux: Ten years later," *ACM SIGARCH Computer Architecture News*, vol. 39, 03 2011.

[4] J. Edge, "The kernel maintainer gap," 10 2013. [Online]. Available: https://lwn.net/Articles/572003/

[5] J. Corbet, "On saying "no"," 10 2013. [Online]. Available: https://lwn.net/Articles/571995/

[6] J. Edge, "On moving on from being a maintainer," 1 2016. [Online]. Available: https://lwn.net/Articles/670087/

[7] J. Corbet, "Two perspectives on the maintainer relationship," 3 2018. [Online]. Available: https://lwn.net/Articles/749676/

[8] D. Vetter, "Maintainers don't scale," 1 2017. [Online]. Available: https://blog.ffwll.ch/2017/01/maintainers-dont-scale.html

[9] J. Edge, "Too many lords, not enough stewards," 1 2018. [Online]. Available: https://lwn.net/Articles/745817/

[10] S. Dean, "The maintainer's paradox: Balancing project and community," 12 2020. [Online]. Available: https://www.linuxfoundation.org/blog/the-maintainers-paradox-balancing-project-and-community/

[11] J. Corbet, "Maintainers truth and fiction," 1 2021. [Online]. Available: https://lwn.net/Articles/842415/

[12] M. S. R. Wen, "What happens when the bazaar grows: a comprehensive study on the contemporary linux kernel development model," Ph.D. dissertation, University of São Paulo, 2021.

[13] E. Pinheiro and P. Meirelles, "Understanding group maintainership model in the linux kernel development," in *Anais do XII Workshop de Visualização, Evolução e Manutenção de Software*. Porto Alegre, RS, Brasil: SBC, 2024, pp. 113–124.

[14] J. Corbet and G. Kroah-Hartman. (2017) Linux kernel development report. [Online]. Available: https://www.linuxfoundation.org/resources/publications/linux-kernel-report-2017?hsLang=en

[15] R. Di Cosmo and S. Zacchiroli, "Software heritage: Why and how to preserve software source code," in *iPRES 2017*, 2017.

[16] A. Pietri, D. Spinellis, and S. Zacchiroli, "The Software Heritage Graph Dataset: Public Software Development Under One Roof," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. Montreal, QC, Canada: IEEE, May 2019, pp. 138–142.

[17] P. Boldi, A. Pietri, S. Vigna, and S. Zacchiroli, "Ultra-large-scale repository analysis via graph compression," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 184–194.

[18] T. Fontana, S. Vigna, and S. Zacchiroli, "Webgraph: The next generation (is in rust)," in *Companion Proceedings of the ACM Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 686–689.

[19] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast? case study on the linux kernel," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 101–110.

[20] M. T. Rahman and P. C. Rigby, "Contrasting development and release stabilization work on the linux kernel," in *International Workshop on Release Engineering*, 2014.

[21] D. Schneider, S. Spurlock, and M. Squire, "Differentiating Communication Styles of Leaders on the Linux Kernel Mailing List," in *Proceedings of the 12th International Symposium on Open Collaboration*. Berlin Germany: ACM, Aug. 2016, pp. 1–10.

[22] S. Ahmed and N. U. Eisty, "Hold on! is my feedback useful? evaluating the usefulness of code review comments," *Empirical Software Engineering*, vol. 30, no. 3, p. 70, Jun. 2025.

[23] M. Hatta, "The role of mailing lists for policy discussions in open source development," *Annals of Business Administrative Science*, vol. 17, no. 1, pp. 31–43, Feb. 2018.

[24] S. Dueñas, V. Cosentino, J. M. Gonzalez-Barahona, A. del Castillo San Felix, D. Izquierdo-Cortazar, L. Cañas-Díaz, and A. Pérez García-Plaza, "Grimoirelab: A toolset for software development analytics," *PeerJ Computer Science*, vol. 7, no. e601, 2021.

[25] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona, "Perceval: software project data at your will," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–4.

[6] See `perceval/backends/core/git.py`, lines 590 to 758: github.com/chaoss/grimoirelab-perceval version 1.3.1

[7] Available at gitlab.softwareheritage.org/swh/devel/docker.git