DUKS: visualizações e análises unificadas para o Kernel Linux

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. *The decentralized nature of the Linux kernel's development hinders the analysis of its maintenance model. Current tools fail to capture nuances such as the flow of commits between repositories. We propose DUKS (Dashboard for Unified Kernel Statistics), a framework that integrates data from mailing lists and git trees collected from the* `Software Heritage` *to provide new insights into the project's health and evolution.*

Resumo. *A natureza descentralizada do desenvolvimento do kernel Linux dificulta a análise de seu modelo de manutenção. Ferramentas atuais não capturam nuances como o fluxo de commits entre repositórios. Propomos o* `DUKS` *(Dashboard for Unified Kernel Statistics), um framework que integra dados de listas de e-mail e árvores git coletadas do* `Software Heritage` *para fornecer novas percepções sobre a saúde e evolução do projeto.*

1. Introdução

O kernel do Linux é uma base da infraestrutura tecnológica moderna, mas seu fluxo de trabalho ainda reflete práticas de desenvolvimento colaborativo de mais de duas décadas, centradas no uso de e-mail [1]. O processo de contribuição baseia-se no envio de *patches* (commits do git) através de listas de e-mail. Nesses canais, contribuidores e mantenedores conduzem um processo iterativo de revisão para garantir a qualidade do código antes de sua integração ao repositório principal (mainline) [Palix et al. 2011]. O crescimento contínuo do kernel em extensão, complexidade e número de contribuidores levanta preocupações sobre a sustentabilidade de seu modelo de manutenção. A comunidade [2] [3] [4] [5] [6] [7] [8]. e a academia alertam que o modelo de desenvolvimento atual pode se tornar insustentável [Wen 2021, Pinheiro and Meirelles 2024]. Um desafio é a dificuldade em estimar o número de mantenedores verdadeiramente ativos. O uso exclusivo do arquivo `MAINTAINERS` é impreciso, e uma medição acurada exige o cruzamento de múltiplas fontes de dados, como listas de e-mail e o histórico do git, uma tarefa inviável sem ferramentas especializadas.

Adicionalmente, técnicas convencionais de mineração de software não capturam as particularidades do ecossistema do kernel. Elas raramente detalham as interações que

[1] Veja `lwn.net/Articles/702177/`
[2] Veja `lwn.net/Articles/572003`
[3] Veja `lwn.net/Articles/571995`
[4] Veja `lwn.net/Articles/670087`
[5] Veja `lwn.net/Articles/749676`
[6] Veja `blog.ffwll.ch/2017/01/maintainers-dont-scale.html`
[7] Veja `lwn.net/Articles/745817`
[8] Veja `lwn.net/Articles/842415`

levaram à aceitação ou rejeição de um patch, ou consideram as fases distintas do ciclo de desenvolvimento, como os períodos de integração (merge) e estabilização [9], que poderiam fornecer *insights* valiosos sobre o fluxo de contribuições.

Neste artigo, propomos o `DUKS` — **Dashboard for Unified Kernel Statistics** [10], uma abordagem inovadora para coletar, agregar e visualizar métricas de desenvolvimento adaptadas ao modelo do kernel do Linux. Nosso método integra dados das listas de e-mail com o grafo de código-fonte e versionamento do projeto `Software Heritage` [Di Cosmo and Zacchiroli 2017]. Apresentamos também uma prova de conceito do `DUKS`, ilustrando o potencial da ferramenta para apoiar análises sobre a saúde do modelo de manutenção do kernel [11].

2. Processamento e Coleta de Dados

Nossa abordagem explora dados públicos do desenvolvimento do Kernel Linux, provenientes de listas de e-mail e repositórios git, para fundamentar análises empíricas e reprodutíveis sobre a sustentabilidade de seu modelo de desenvolvimento.

A primeira etapa consiste na coleta e estruturação de dados das listas de e-mail do kernel, como as arquivadas no *Kernel Lore Archives*. Desenvolvemos um esquema de dados especializado para capturar o conteúdo, os metadados dos *patches* e informações relevantes extraídas do corpo e assunto dos e-mails (*e.g.,* versão do *patchset*, contagem de *patches*). Os dados consolidados são armazenados em um formato analítico aberto, como o Apache Parquet, que permite o particionamento eficiente para análises temporais.

A segunda etapa utiliza o histórico de código-fonte arquivado pela iniciativa `Software Heritage`. O projeto organiza o histórico de múltiplos repositórios em um único grafo de Merkle deduplicado, onde as entidades são identificadas por *hashs* estáveis, até quando originadas de repositórios distintos [Pietri et al. 2019], propriedade fundamental para navegar pelo histórico de desenvolvimento do kernel.

A estratégia central é a integração dos dados das listas de e-mail com o grafo do `Software Heritage`. Ao vincular as discussões com os artefatos de código correspondentes, superamos as limitações de uma análise restrita ao histórico git. Essa abordagem unificada permite investigar a autoria dos *patches*, o processo de revisão, e a relação entre as discussões e os commits aplicados nos diferentes repositórios do kernel.

Para nossa **prova de conceito** apresentada neste trabalho, selecionamos a exportação do grafo comprimido do `Software Heritage` [12]. Devido a limitações de armazenamento, utilizamos o subgrafo de 1.5TiB *"History and hosting" Compressed graph* [Boldi et al. 2020]. O subgrafo contém revisões (*commits*) e origens (repositórios), mas não o código-fonte. A partir da URL do repositório, buscamos o *snapshot* mais recente no grafo do `Software Heritage` para extrair as *releases* (*tags*) e a última

[9] Veja `linuxfoundation.org/resources/publications/linux-kernel-report-2017`

[10] Veja `github.com/linux-duks/DUKS`

[11] O pacote de replicação está disponível em `archive.softwareheritage.org/swh:1:rev:dbce519d25dcaa8cc1405098ce2f20fd44f02636;origin=https://github.com/linux-duks/DUKS-2025-replication-pkg`

[12] Exportação de 18/05/2025: `docs.softwareheritage.org/devel/swh-export/index.html`

revisão.

Em seguida, percorremos todo o histórico de commits via busca em largura, a partir da revisão mais recente, criando um *dataset* tabular com data, hash e assinaturas de cada commit. Para cruzar esses dados com as listas de e-mail e o arquivo `MAINTAINERS`, revertemos a pseudonimização do `Software Heritage`, complementando nosso *dataset* com os dados de autoria obtidos diretamente dos repositórios git originais.

Para obter uma lista de mantenedores oficiais, obtivemos as entradas do arquivo `MAINTAINERS` do Linux para cada revisão de arquivo. Juntamos esses dados ao nosso conjunto usando o `DuckDB` [13]. Em seguida, usando a biblioteca `Polars` [14], agregamos os commits por data de criação para formar uma série temporal das métricas. Analisamos as atribuições de cada mensagem de commit, identificando quais dos contribuidores marcados também estão no arquivo `MAINTAINERS`.

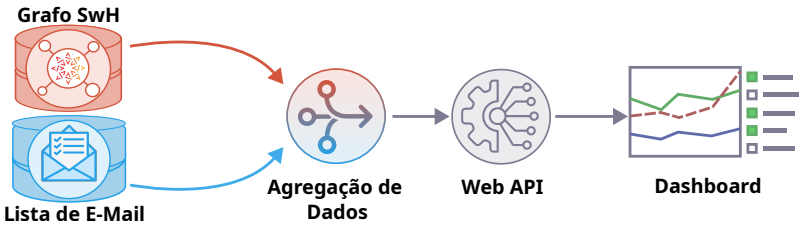

Figura 1. Visão em Alto Nível dos Componentes Arquiteturais do DUKS

A Figura 1 ilustra uma visão geral da aplicação. Usamos ferramentas fornecidas pelo `Software Heritage`, como a `swh.graph` [15] implementada em Rust. Nosso próximo incremento à prova de conceito é desenvolver uma camada de compatibilidade para acessar as entradas das listas de e-mail (em azul da figura) e cruzá-las com as múltiplas árvores git do grafo original do `Software Heritage` (em laranja). As métricas são pré-calculadas antes de serem servidas para apresentação pelo *Dashboard*.

3. Um Dashboard para Estatísticas Unificadas do Kernel

Na primeira versão do `DUKS`, apresentamos uma visualização de séries temporais focada na atividade e na carga de trabalho dos mantenedores. Todos os dados utilizados são provenientes do `Software Heritage`, abordando o repositório *mainline*.

A estratégia descrita na Seção 2 fornece uma linha do tempo de todas as contribuições observadas no repositório. No entanto, a data do commit de um *patch* aceito representa somente o passo final no processo de contribuição. De acordo com Jiang *et al.* [Jiang et al. 2013], a maioria dos *patches* leva de três a seis meses para ser integrada. Essa média pode mudar dependendo da complexidade de cada contribuição. Isso destaca quanto de esforço do contribuidor ocorre antes da data registrada do commit. Portanto, ao analisar o envolvimento de um contribuidor, cada commit aceito reflete a culminação de potencialmente meses de trabalho e discussão. Para levar isso em conta, aplicamos uma contagem móvel a várias métricas, estipulando atividade de contribuidores nos períodos que antecederam cada commit.

[13] Veja `duckdb.org/`
[14] Veja `pola.rs/`
[15] Veja `docs.softwareheritage.org/devel/apidoc/swh.graph.html`

Uma das perguntas iniciais que motivaram nosso trabalho foi: **Quantos mantenedores ativos existem no kernel do Linux?** Como mencionado anteriormente, o arquivo `MAINTAINERS` lista os responsáveis por cada *driver* e subsistema do kernel. Na Figura 2 analisamos essa questão em detalhes. Utilizamos janelas de um mês ou um ano para todas as séries temporais, exceto para as *dados do arquivo* `MAINTAINERS`, que representa um valor fixo em um determinado ponto no tempo.

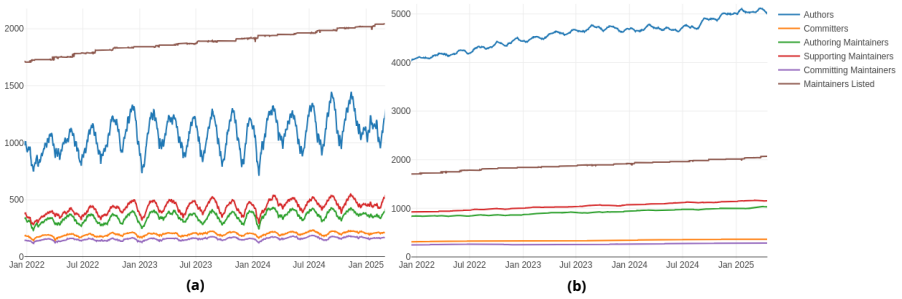

Figura 2. Atividade de Mantenedores e Contribuidores em diferentes papéis

Os gráficos na Figura 2 exibem, em ordem decrescente de magnitude: *Maintainers Listed* (número de mantenedores registrados no arquivo `MAINTAINERS`); *Authors* (número de autores de commits); *Supporting Maintainers* (número de mantenedores atuando em papéis diferentes de *Autor* ou *Committer*); *Authoring Maintainers* (número de autores também listados como mantenedores); *Committers* (número de indivíduos que criaram commits); e *Committing Maintainers* (número de mantenedores que criaram commits).

O gráfico (Figura 2.a) apoia nossa suspeita de uma disparidade entre mantenedores listados e ativos. Essa diferença permanece considerável mesmo considerando janelas móveis de um ano (Figura 2.b). Embora essa disparidade não seja necessariamente um sinal de preocupação, dado que muitas áreas do kernel do Linux são estáveis e requerem pouca ou nenhuma manutenção, a lacuna persistente entre as séries de *Committers* e *Mantenedores que Fizeram Commits* (as duas últimas) indica que nem todos os indivíduos que fazem commit de código estão formalmente listados no arquivo `MAINTAINERS`.

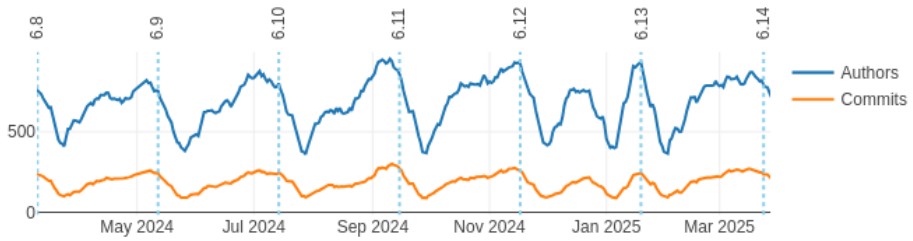

Figura 3. Efeitos do Ciclo de Lançamento/Estabilização nas Contribuições

Também investigamos as contribuições no ciclo de lançamento/estabilização, inspirados por Rahman *et al.* [Rahman and Rigby 2014]. Na Figura 3, mostramos a média de commits e autores únicos em uma janela de duas semanas. Conforme explorado pelo autor anterior, o impacto do método de gerenciamento centrado em datas é visível.

4. Discussão e Considerações Finais

O fluxo de trabalho de desenvolvimento do kernel do Linux é predominantemente baseado em discussões e revisões em suas listas de e-mail. Idealizamos o `DUKS`, uma ferramenta inovadora de análise de repositórios adaptada para o kernel do Linux. Ele incorpora dados de suas listas de e-mail ao modelo de grafo usado para representar repositórios de código no `Software Heritage`, permitindo análises anteriormente inviáveis.

Estudos recentes exploram o valor científico dos dados extraídos das listas de e-mail do kernel do Linux. Por exemplo, Schneider *et al.* [Schneider et al. 2016] analisam as diferenças na forma como os líderes se comunicam nas listas de e-mail do kernel; e Hatta *et al.* [Hatta 2018] examinam o papel das listas de e-mail no desenvolvimento de Software Livre, focando no projeto Debian.

Além disso, o GrimoireLab é um conjunto de ferramentas para recuperar, enriquecer e visualizar dados sobre o desenvolvimento de software [Dueñas et al. 2021]. É uma ferramenta poderosa de propósito geral, mas não adaptada ao kernel do Linux. Embora alguns dos propósitos do GrimoireLab se sobreponham aos nossos, nosso trabalho considera detalhes específicos do kernel do Linux. Por exemplo, para identificar contribuidores, utilizamos o arquivo `MAINTAINERS` (exclusivo do kernel do Linux), bem como assinaturas de atribuição, também conhecidas como *git trailers*. No entanto, a ferramenta coletora do GrimoireLab, Perceval [Dueñas et al. 2018], descarta todos os trailers não padronizados [16].

Para nossa prova de conceito, criamos nosso próprio índice do `Software Heritage` incluindo **apenas** os repositórios relevantes para o desenvolvimento do kernel do Linux. Essa abordagem elimina a necessidade de máquinas de alto desempenho e armazenamento para lidar com o grafo completo fornecido pelo projeto. Adicionalmente, usamos uma pilha auto-hospedada configurada via `swh-docker` [17]. Com essa configuração, por exemplo, enfileiramos tarefas de indexação para os repositórios listados no arquivo `MAINTAINERS`, que correspondem aos subsistemas, drivers e árvores de ferramentas do projeto. Embora essa visão de grafo não seja ideal para analisar métricas de séries temporais, ela permite diferentes estratégias de exploração, algoritmos e uma nova classe de visualizações baseadas em grafos. No entanto, ainda é possível derivar estruturas de séries temporais do grafo, como demonstrado em nossa prova de conceito.

Nenhuma mensagem de commit isolada reflete, por si só, o tempo e o esforço que um contribuidor investe em um *patch*. As listas de e-mail podem ajudar a revelar esse esforço, preservando dados que possibilitam identificar múltiplas versões da mesma contribuição, mapear discussões técnicas que moldaram decisões de design específicas e entender quais membros da comunidade contribuíram ao longo do desenvolvimento do *patch*. Ao integrar fontes complementares de informação e considerar as particularidades do Kernel Linux, nossa abordagem visa a facilitar a compreensão de seu fluxo de contribuição. Consequentemente, nossa abordagem apoia múltiplas análises sobre a sustentabilidade do modelo de manutenção adotado por um dos sistemas de software mais críticos na computação moderna.

[16] Veja *perceval/backends/core/git.py*, linhas 590 a 758: `github.com/chaoss/grimoirelab-perceval/tree/1.3.1`

[17] Veja `gitlab.softwareheritage.org/swh/devel/docker.git`

Referências

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

Di Cosmo, R. and Zacchiroli, S. (2017). Software heritage: Why and how to preserve software source code. In *iPRES 2017*.

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

Dueñas, S., Cosentino, V., Gonzalez-Barahona, J. M., del Castillo San Felix, A., Izquierdo-Cortazar, D., Cañas-Díaz, L., and Pérez García-Plaza, A. (2021). Grimoirelab: A toolset for software development analytics. *PeerJ Computer Science*, 7(e601).

Hatta, M. (2018). The role of mailing lists for policy discussions in open source development. *Annals of Business Administrative Science*, 17(1):31–43.

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

Palix, N., Saha, S., Thomas, G., Calvès, C., Lawall, J., and Muller, G. (2011). Faults in linux: Ten years later. *ACM SIGARCH Computer Architecture News*, 39.

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

Pinheiro, E. and Meirelles, P. (2024). Understanding group maintainership model in the linux kernel development. In *Anais do XII Workshop de Visualização, Evolução e Manutenção de Software*, pages 113–124, Porto Alegre, RS, Brasil. SBC.

Rahman, M. T. and Rigby, P. C. (2014). Contrasting development and release stabilization work on the linux kernel. In *International Workshop on Release Engineering*.

Schneider, D., Spurlock, S., and Squire, M. (2016). Differentiating Communication Styles of Leaders on the Linux Kernel Mailing List. In *Proceedings of the 12th International Symposium on Open Collaboration*, pages 1–10, Berlin Germany. ACM.

Wen, M. S. R. (2021). *What happens when the bazaar grows: a comprehensive study on the contemporary Linux kernel development model*. PhD thesis, University of São Paulo.