Estratégias de ensino para incentivar a participação consistente em projetos de Software Livre

David Tadokoro, Rafael Passos, Paulo Meirelles

Centro de Competência em Software Livre
Instituto de Matemática e Estatística
Universidade de São Paulo, Brasil
{*davidbtadokoro,rcpassos*}*@usp.br,paulormm@ime.usp.br*

Abstract. *Free software development is essential for governments, companies, and society, sustained by communities that maintain the software and define its evolution rules. A recurring challenge is renewing the workforce by forming continuous contributors beyond occasional and isolated contributions. This work presents strategies to train new developers. We applied an approach in a one-semester course in which students: (1) learned Linux kernel fundamentals and submitted patches; (2) contributed to GNU/Linux ecosystem tools; (3) packaged software for the Debian distribution; and (4) collaborated on a project of their choice, exploring different forms of engagement. Our approach reproduces the trajectory of a "self-taught" contributor in a focused and immersive environment, with workshops, mentoring, and close guidance, simulating a free software community. These strategies proved effective by providing fast feedback and fostering progressive learning. The results show greater student confidence for continuous contributions and developing technical and interpersonal skills relevant to any project: such strategies can rapidly prepare dozens of developers with a solid foundation, ready to engage sustainably in free software ecosystems.*

Resumo. *O desenvolvimento de software livre é essencial para governos, empresas e sociedade, sustentado por comunidades que mantêm o software e definem suas regras de evolução. Um desafio recorrente é renovar a força de trabalho, formando contribuidores contínuos, indo além das contribuições ocasionais e isoladas. Este trabalho apresenta estratégias para treinar novos desenvolvedores. Aplicamos a abordagem em uma disciplina de um semestre, na qual os estudantes: (1) aprenderam fundamentos do kernel Linux e enviaram patches; (2) contribuíram com ferramentas do ecossistema GNU/Linux; (3) empacotaram software para a distribuição Debian; e (4) colaboraram em um projeto de sua escolha, explorando diferentes formas de engajamento. Nossa abordagem reproduz a trajetória de um contribuidor "autodidata", mas em um ambiente focado e imersivo, com oficinas, mentoria e acompanhamento próximo, simulando uma comunidade de software livre. Essas estratégias mostraram-se eficientes ao oferecer retorno rápido e favorecer o aprendizado progressivo. Os resultados mostram maior confiança dos estudantes para contribuições contínuas e para o desenvolvimento de habilidades técnicas e interpessoais relevantes a qualquer projeto: tais estratégias podem rapidamente preparar dezenas de desenvolvedores com uma base sólida, prontos para atuar de forma sustentável em ecossistemas de software livre.*

1. Introdução

Projetos de software livre geralmente dependem de uma comunidade de desenvolvedores composta por pessoas que contribuem com alterações de código, relatos de *bugs*, testes e outras atividades, denominadas **contribuidores**; e por desenvolvedores que mantêm o projeto por meio da revisão e integração de alterações, definição de diretrizes de desenvolvimento, entre outras tarefas, denominados **mantenedores**. Nesse contexto, a sustentabilidade de qualquer projeto de software livre depende da renovação de sua força de trabalho, a partir da chegada de novos contribuidores, conhecidos como **novatos**.

Diversos estudos investigaram as barreiras de entrada enfrentadas por novatos no ecossistema de software livre. Por exemplo, utilizando um modelo de barreiras de entrada, [Steinmacher et al. 2015] analisaram formas de aumentar a autoeficácia percebida dos novatos em suas primeiras contribuições, abordando aspectos relacionados à motivação e à confiança. Em dois estudos, [Balali et al. 2018] e [Balali et al. 2020] investigaram a mentoria como abordagem educacional para introduzir novatos ao software livre, discutindo os desafios enfrentados por mentores e mentorados e apresentando estratégias para enfrentá-los. Em pesquisas mais recentes,[Tan et al. 2023] e [Xiao et al. 2022] apuraram como tarefas recomendadas a novatos, frequentemente chamadas de *Good First Issues*, podem promover a entrada de novos contribuidores, bem como estratégias para esse processo.

No entanto, tais trabalhos se atém ao que é chamado na literatura de *one-time contributors* (contribuidores pontuais), isto é, contribuidores que realizam apenas uma contribuição, geralmente simples e de baixo impacto. Em muitos casos, essas contribuições demandam mais tempo dos mantenedores no processo de orientação do que oferecem em retorno. Impactos mais relevantes surgem quando os novatos continuam contribuindo e superam o estágio de contribuição pontual. Além disso, embora valiosos para entender e propor melhorias no processo de entrada de novatos, estes estudos tratam de aspectos específicos e isolados da trajetória de contribuidores em software livre.

Explorando as lacunas presentes neste contexto, este trabalho apresenta um conjunto de estratégias voltadas ao treinamento de desenvolvedores com potencial para se tornarem contribuidores consistentes (e de alto impacto) na maioria dos projetos de software livre, por meio do desenvolvimento de habilidades específicas e da aquisição de experiência prática. Estas estratégias são apresentadas na forma de um programa de mentoria com duração de vários meses, que são fruto de uma disciplina universitária focada no desenvolvimento de software livre, com ênfase no projeto do *kernel* Linux e seu ecossistema.

A partir de apoio contínuo de mentores (monitores) experientes, oficinas presenciais, materiais didáticos e mecanismos leves de acompanhamento, simulamos uma comunidade de software livre centrada em formar contribuidores capazes de se tornarem participantes ativos e frequentes. Nesse sentido, as estratégias apresentadas representam uma abordagem alternativa para fomentar contribuidores duradouros no ecossistema de software livre.

O restante deste trabalho está estruturado da seguinte forma: a Seção 2 descreve a experiência da disciplina universitária e os métodos utilizados para coleta e análise de dados; a Seção 3 apresenta os resultados obtidos e discute sua relação com a efetividade

das estratégias e suas limitações; a Seção 4 expõe as estratégias de treinamento de forma direta e sintética para referência futura; e a Seção 5 conclui o trabalho com oportunidades para estudos futuros.

2. Metodologia

Esta seção descreve os recursos utilizados na experiência de treinamento dos estudantes, a abordagem empregada durante a disciplina e os métodos utilizados para coletar e analisar dados.

2.1. Recursos Utilizados

A condução do treinamento exigiu recursos físicos e humanos, materiais didáticos e mecanismos de acompanhamento. Palestras tradicionais e oficinas presenciais foram elementos centrais, requerendo um espaço semanal com capacidade para trinta participantes, equipado com mesas e acesso à Internet de alta velocidade, adequado tanto para exposições quanto para atividades em grupo. Participaram da experiência 3 mentores e um grupo de 27 desenvolvedores em treinamento. Materiais didáticos, como apresentações, tutoriais e textos de referência, foram produzidos ou disponibilizados pelos mentores ao longo do treinamento. Por fim, mecanismos simples de monitoramento de progresso foram implementados, como o registro constante de atividades e apresentações rápidas realizadas pelos participantes.

2.2. Estratégia para o Treinamento dos Estudantes

O treinamento foi implementado ao longo de uma disciplina universitária com duração de quatro meses, envolvendo 21 estudantes de graduação e 6 de pós-graduação, com perfis acadêmicos diversos. A descrição da abordagem busca se manter o mais independente possível desse contexto específico, de modo a facilitar sua reprodutibilidade.

A proposta visou inserir os participantes em diferentes modelos e comunidades de desenvolvimento de software livre. O programa da disciplina foi pensado para utilizar os projetos do *kernel* Linux e da distribuição Debian como estudos de caso, estruturando-se em quatro fases:

Fase 1 – Desenvolvimento do *kernel* Linux: Os participantes iniciaram com tutoriais[1] para aprender os fundamentos do desenvolvimento do *kernel* Linux, como a configuração do ambiente, fluxos de trabalho de desenvolvimento e anatomia de *drivers*, no contexto de um subsistema específico. A fase foi concluída com o envio de *patches*, primeiro validados por mentores e, posteriormente, submetidos como contribuições reais ao projeto.

Fase 2 – Projeto de apoio ao Linux: Após a experiência inicial com o modelo baseado em listas de e-mail, os participantes passaram a contribuir com um projeto hospedado na *Web* (no *GitHub*) de apoio ao ecossistema Linux; foi utilizada a ferramenta *kwork-flow* [Siqueira, Rodrigo and Tadokoro, David and Tavares, Matheus 2025].

Fase 3 – Empacotamento Debian: Mantendo-se no ecossistema Linux, os participantes passaram a trabalhar com empacotamento de software na distribuição Debian, uma camada mais próxima dos usuários finais em comparação ao desenvolvimento do *kernel*.

[1] `flusp.ime.usp.br/kernel`

Fase 4 – Contribuições independentes a projetos de software livre: Na fase final, os participantes escolheram livremente projetos de software livre para os quais contribuíram de forma independente.

Ao longo de todas as fases, foram realizadas oficinas presenciais, tanto aquelas voltadas a atividades comuns a todos os participantes (como os tutoriais da Fase 1 e as oficinas de empacotamento da Fase 3), quanto aquelas que ofereciam suporte ao trabalho individual com orientação dos mentores (como na Fase 2 e na Fase 4).

A intenção era simular a dinâmica de uma comunidade de software livre, na qual participantes inexperientes evoluem por meio da interação com mentores, que assumem o papel de mantenedores experientes, inclusive atuando como tal em situações reais, como na revisão de *patches* antes da submissão que encerrou a Fase 1. No entanto, dada a proximidade física e a disponibilidade dos mentores, a troca de conhecimento pôde ser mais eficiente e adaptada às necessidades individuais.

Nas três primeiras fases, a curadoria das sugestões de contribuição foi considerada fundamental. As tarefas deveriam ser acessíveis a iniciantes, mas com substância suficiente para promover engajamento e compreensão. Tarefas triviais ou excessivamente simples foram desencorajadas para favorecer o aprendizado significativo.

As aulas, sessões expositivas e oficinas foram realizadas de forma presencial. A abordagem adotada simulou a dinâmica de comunidades reais, com mentores atuando como mantenedores que orientam, instruem e ocasionalmente revisam contribuições dos participantes. A interação próxima permitiu *feedback* eficiente e adaptativo.

2.3. Coleta e Análise de Dados

Para avaliar a efetividade da abordagem, foram coletados dados qualitativos a partir de três fontes:

Relatos em *blogs*: Os participantes documentaram seu progresso ao longo das fases por meio de postagens em *blogs*, simulando registros de atividades técnicas.

Questionário ao final do treinamento: Um questionário com 47 perguntas coletou dados demográficos, percepções sobre os aprendizados e *feedback* sobre a estrutura da disciplina. As perguntas incluíram escalas de Likert, questões fechadas (sim/não) e perguntas abertas.

Entrevistas com mentores: Foram realizadas entrevistas com os mentores para registrar suas observações sobre o processo de treinamento e a evolução dos participantes.

Os dados do questionário foram agregados e categorizados (positivos, neutros ou negativos) para as principais questões. As postagens e entrevistas foram analisadas qualitativamente, buscando padrões recorrentes e insights relevantes.

3. Resultados e Discussão

Esta seção apresenta os principais resultados advindos da análise dos dados e discute como eles corroboram a efetividade das diretrizes de treinamento.

3.1. Principais Observações

A abordagem adotada conseguiu alterar a percepção dos participantes e desenvolver habilidades fundamentais para a contribuição em projetos de software livre, mesmo entre aqueles sem experiência prévia. Alguns destaques incluem:

Mudança de percepção e aumento da confiança. Embora a maioria estivesse familiarizada com o conceito de software livre, apenas um quarto havia contribuído anteriormente. Ao final do treinamento, cerca de 75% relataram sentir-se mais aptos e confiantes para contribuir com qualquer projeto de software livre, o que indica evolução perceptível.

Desenvolvimento técnico: Dificuldades iniciais com ferramentas de linha de comando e ambientes Linux foram comuns, mas foram superadas gradualmente. Cerca de 90% relataram melhora em sua proficiência com Git, e 85% reconheceram essa habilidade como essencial para contribuir com software livre. A experiência prática com fluxos de trabalho baseados em e-mail e plataformas *Web* contribuiu para uma maior familiaridade com os conceitos centrais do desenvolvimento de software livre.

Mentoria e oficinas: A atuação dos mentores e as oficinas presenciais foram aspectos centrais. Mais de 85% dos participantes apontaram essas atividades como críticas para seu aprendizado e evolução enquanto contribuidores. Com o suporte próximo dos mentores, muitos passaram da dependência à autonomia ao longo do processo, conseguindo realizar contribuições reais e completar as atividades com pouca orientação ao final.

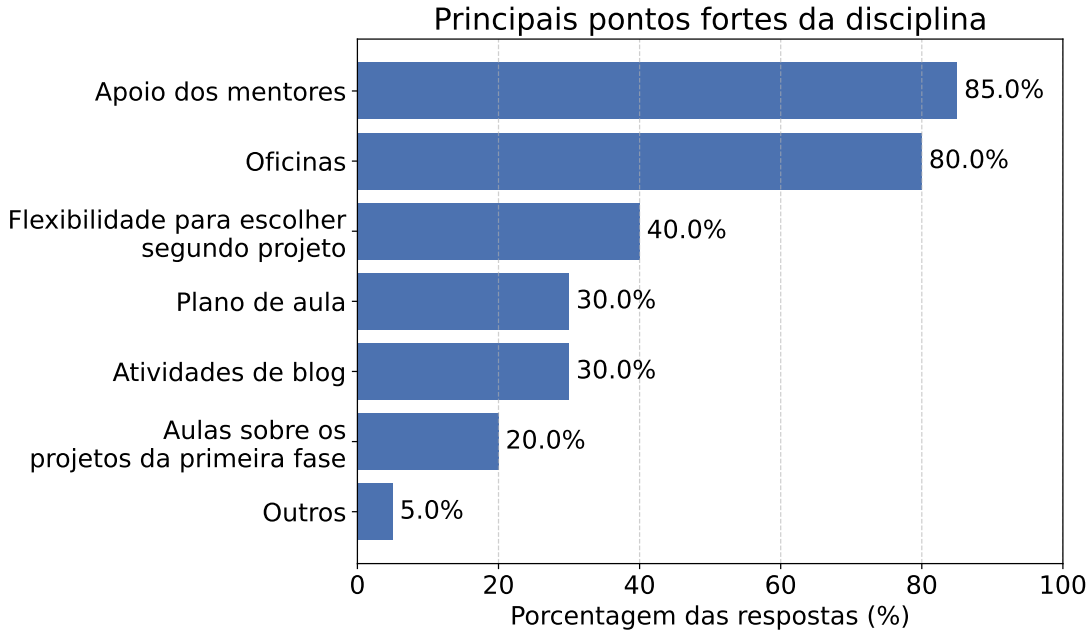

Figura 1. Gráfico de barras com os principais pontos fortes da disciplina segundo os participantes.

Observações gerais: A Figura 1 resume os pontos fortes da disciplina segundo a perspectiva dos participantes, reforçando a relevância da mentoria próxima e das sessões presenciais. Todos os participantes realizaram ao menos quatro contribuições para quatro projetos diferentes de software livre em um período relativamente curto, o que lhes proporcionou uma visão ampla sobre diferentes modelos de desenvolvimento. Em vez de repetir o papel de contribuidores pontuais, os participantes foram incentivados a desenvolver tanto habilidades técnicas

quanto interpessoais relevantes, visando contribuições mais consistentes e significativas. Iniciar o treinamento com o desenvolvimento no *kernel* Linux representou um desafio inicial para a maioria. No entanto, a experiência foi benéfica a longo prazo, pois incentivou a resolução autônoma de problemas e aprofundou a compreensão sobre fluxos e dinâmicas centrais de projetos de software livre. Materiais didáticos bem planejados também foram essenciais para o sucesso do treinamento. Embora alguns obstáculos tenham sido atribuídos a falhas nos tutoriais, a maior parte do conhecimento foi transmitida por meio desses materiais, reforçando sua importância.

3.2. Limitações

Algumas limitações dos métodos utilizados neste estudo podem afetar sua validade e aplicabilidade.

A dinâmica aluno-professor de uma disciplina universitária, associada à atribuição de notas, pode introduzir viés no nível de dedicação dos participantes. Os estudantes podem sentir-se motivados a investir mais esforço, mas também podem optar por realizar o mínimo necessário para concluir a disciplina. Ainda assim, a motivação é um fator-chave para a permanência como contribuidor em projetos de software livre, e este trabalho não tem como objetivo analisar esse aspecto em profundidade.

Por outro lado, as diretrizes apresentadas estão fortemente relacionadas ao ecossistema Linux. Embora os resultados sugiram que essa escolha tenha produzido efeitos positivos, outros conjuntos de projetos de software livre podem ser igualmente eficazes ou gerar resultados distintos, ainda que relevantes.

Por fim, embora este trabalho se concentre em características comuns a projetos de software livre, cada projeto possui particularidades técnicas, como tecnologias específicas e conhecimento de domínio. Esses aspectos precisam ser enfrentados em algum momento da trajetória de contribuição, mas não são abordados pelas diretrizes apresentadas neste trabalho.

4. Diretrizes para o Treinamento de Contribuidores Consistentes

Esta seção apresenta diretrizes práticas para orientar e capacitar pessoas a se tornarem contribuidoras autônomas, confiantes e duradouras em projetos de software livre. As diretrizes baseiam-se na metodologia aplicada durante uma disciplina universitária de quatro meses, descrita na Seção 2, e são informadas pelos resultados e reflexões discutidos na Seção 3. As subseções a seguir abordam os recursos necessários, a estrutura do treinamento e suas fases, recomendações de acompanhamento e observações finais.

4.1. Recursos Necessários

A implementação das diretrizes exige um conjunto mínimo, porém intencional, de recursos físicos, humanos e informacionais. Esses recursos estão resumidos na Tabela 1.

4.2. Estrutura e Fases do Treinamento

O treinamento segue uma estrutura progressiva e imersiva que expõe os participantes a diferentes ecossistemas e modelos de desenvolvimento de software livre. Para apoiar essa progressão, o treinamento é dividido em fases sequenciais, embora sobreposições

possam ocorrer (por exemplo, a continuidade de uma contribuição iniciada em uma fase anterior). Cada fase desenvolve habilidades técnicas (*hard skills*) e interpessoais (*soft skills*), fortalecendo gradualmente a autonomia e o engajamento com a comunidade.

Tabela 1. Recursos necessários para o treinamento de contribuidores consistentes em projeto de software livre

Tipo de Recurso	Descrição
Espaço físico	Sala disponível semanalmente com capacidade para vinte a trinta pessoas, mesas para trabalho em grupo e acesso à Internet de alta velocidade.
Mentores	Pessoas com experiência em software livre, capazes de orientar e apoiar os participantes, além de revisar contribuições. Idealmente, um mentor para cada dez participantes.
Participantes	Desenvolvedores com conhecimentos básicos em programação e Git; não é exigida experiência prévia com software livre.
Materiais didáticos	Tutoriais, apresentações e documentação de referência fornecidos e mantidos pelos mentores.
Ferramentas de acompanhamento	Mecanismos leves como entregas semanais e apresentações presenciais curtas para monitorar o progresso.

Não prescrevemos uma duração fixa, mas recomendamos programas com duração de três meses a um semestre acadêmico. Esse intervalo permite a assimilação dos conceitos e o desenvolvimento das experiências sem sobrecarregar os participantes, mantendo o engajamento ao longo do tempo. Nesse sentido, sugerimos a seguinte estrutura de fases com durações aproximadas.

Fase Inicial (2 meses): Esta fase promove disciplina e autonomia ao introduzir os participantes a conceitos fundamentais de software livre, como a natureza das contribuições, formas de submissão e participação no processo de revisão. Recomenda-se o uso de um modelo de desenvolvimento baseado em e-mails, como o utilizado pelo *kernel* Linux ou pelo projeto Git, para familiarizar os participantes com esse fluxo de trabalho menos intuitivo. Esse modelo expõe os participantes a práticas reais: geração e envio de *patches*, recebimento de *feedback* por e-mail, interpretação de revisões *inline*, entre outras. No caso do *kernel* Linux, sugere-se focar em um subsistema com comunidade ativa e acolhedora, e, idealmente, com um mantenedor experiente disposto a apoiar o programa. A fase deve ser concluída com o envio de contribuições primeiramente para revisão simulada pelos mentores, seguido da submissão oficial aos mantenedores e à comunidade.

Fase de Contribuições Independentes (2 meses): Após a fase estruturada inicial, os participantes devem aplicar suas habilidades de forma independente em outro projeto de software livre. Por "independente", entende-se que os mentores não irão selecionar tarefas, revisar previamente as contribuições ou escolher o projeto. No entanto, devem permanecer disponíveis para suporte geral por meio de materiais didáticos, oficinas e orientação. Embora os projetos não precisem ser predefinidos,

os mentores devem validar a adequação dos projetos escolhidos. Recomenda-se preparar previamente uma lista de sugestões. A única exigência é que os projetos sejam baseados em plataformas *Web* (por exemplo, hospedados no GitHub ou GitLab).

Fase Complementar de Curta Duração (2 semanas): Uma experiência breve (idealmente realizada em duas sessões de oficina) pode enriquecer a compreensão dos participantes sobre o desenvolvimento de software livre. Essa fase deve focar em um único projeto ou comunidade, oferecendo uma imersão sem grandes exigências técnicas. Recomenda-se fortemente a atividade de empacotamento Debian, pois permite aos participantes realizarem tarefas simples e obterem uma visão ampla do ecossistema com baixo custo de entrada.

Por fim, recomenda-se a realização do programa em sessões semanais de quatro horas, com intervalo de vinte minutos após as duas primeiras horas. Embora as sessões possam ser divididas em dois dias, a experiência mostra que uma única sessão mantém o engajamento e evita o custo de chaveamento de contexto. As sessões devem ser presenciais e incluir oficinas, aulas expositivas e outras dinâmicas didáticas conforme necessário. Naturalmente, as atividades se estendem além das sessões, exigindo estudo dos materiais, planejamento, produção e revisão das contribuições por parte dos participantes.

4.3. Recomendações de Acompanhamento

Para garantir progresso contínuo e, ao mesmo tempo, promover autonomia, os mentores devem implementar mecanismos de acompanhamento adaptados ao contexto específico. Ainda assim, apresentamos as seguintes recomendações:

Registro de progresso individual: Os participantes devem manter *blogs* ou registros que documentem suas atividades, reflexões e dificuldades enfrentadas.

Revisões na fase inicial: Todas as contribuições da fase inicial devem ser revisadas internamente antes de serem submetidas publicamente.

Apresentações periódicas: Utilizando o formato de apresentação curta (aproximadamente sete minutos), os participantes devem relatar mensalmente seu progresso.

Acompanhamento informal nas oficinas: Durante as oficinas presenciais, os mentores devem observar discretamente os participantes, identificando quem está mais avançado e quem pode estar com dificuldades.

4.4. Observações Importantes

Diversos aspectos são fundamentais para o sucesso no treinamento de contribuidores consistentes:

Desafie cedo, não tarde: Iniciar com um projeto tecnicamente exigente, como o *kernel* Linux, embora possa parecer intimidador, acelera a aprendizagem e desenvolve resiliência.

Priorizar a mentoria: A mentoria próxima e presencial é essencial para simular um ambiente de alto retorno de *feedback*, como o das comunidades de software livre, acelerando o processo de aprendizagem em comparação com comunidades online tipicamente assíncronas.

Curadoria de tarefas: As contribuições devem ser cuidadosamente selecionadas, suficientemente desafiadoras para promover crescimento, mas acessíveis o bastante para permitir o engajamento significativo de iniciantes.

Foco em habilidades essenciais: É importante enfatizar habilidades amplamente aplicáveis, como o uso do Git, colaboração por e-mail e em plataformas *Web*, comunicação clara e etiqueta em comunidades.

Simular a comunidade: Oficinas e mentoria devem reproduzir as dinâmicas interpessoais típicas das comunidades de software livre, mas com maior clareza e retorno de *feedback*.

5. Conclusão

Reconhecendo a escassez de estudos sobre métodos voltados à formação de contribuidores consistentes em software livre, para além das contribuições pontuais, este estudo apresenta uma proposta de treinamento com esse foco. A abordagem é apresentada na forma de diretrizes de treinamento e surgiu a partir de uma experiência universitária de quatro meses. Os resultados indicam que a aplicação dessas diretrizes pode: (1) aumentar a confiança dos participantes para contribuir com qualquer projeto de software livre e (2) aprimorar habilidades técnicas e interpessoais fundamentais, com ampla aplicabilidade na maioria dos projetos.

A análise dos resultados também destacou a importância do suporte próximo de mentores e da convivência presencial com colegas e mentores durante oficinas, o que simula o ambiente de uma comunidade de software livre, ainda que com foco mais centrado nas necessidades dos aprendizes.

Como trabalhos futuros, pretende-se testar variações das diretrizes com diferentes conjuntos de projetos de software livre. A repetição da experiência, com ampliação da coleta de dados, pode permitir seu refinamento e avaliação mais precisa de impactos e limitações. Entre os dados relevantes estão: autoavaliação das habilidades técnicas dos participantes (por exemplo, uso de Git e linha de comando) antes e depois do treinamento; perfil demográfico (idade e proficiência em línguas); e se, e como, continuaram contribuindo com software livre, cuidando para evitar viés amostral, pois os que continuam tendem a responder mais.

Reprodutibilidade e Disponibilidade dos Dados

O estudo apresentado neste artigo baseia-se em um trabalho em andamento, portanto, organizamos o subconjunto dos dados coletados necessário para a reprodução dos resultados apresentados na Seção 3. Esse subconjunto, juntamente com outros artefatos, está armazenado em um repositório no Zenodo [2].

Agradecimentos

Este estudo foi financiado, em parte, pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Código de Financiamento 001), pela Universidade de São Paulo – USP (Proc. 22.1.9345.1.2), pela Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP e pela Fundação Sistema Estadual de Análise de Dados – SEADE (Proc. 2023/18026-8). Agradecemos também aos estudantes da disciplina MAC0470/5856 – Desenvolvimento de Software Livre, do IME-USP, por suas contribuições. Em especial, agradecemos aos materias, apoio e orientações dos desenvolvedores do *kernel* Linux, Rodrigo Siqueira e Marcelo Schmitt.

[2] doi.org/10.5281/zenodo.16809638

Referências

Balali, S., Annamalai, U., Padala, H. S., Trinkenreich, B., Gerosa, M. A., Steinmacher, I., and Sarma, A. (2020). Recommending tasks to newcomers in oss projects: How do mentors handle it? In *Proceedings of the 16th International Symposium on Open Collaboration*, OpenSym '20, New York, NY, USA. Association for Computing Machinery.

Balali, S., Steinmacher, I., Annamalai, U., Sarma, A., and Gerosa, M. A. (2018). Newcomers' barriers. . . is that all? an analysis of mentors' and newcomers' barriers in oss projects. *Comput. Supported Coop. Work*, 27(3–6):679–714.

Siqueira, Rodrigo and Tadokoro, David and Tavares, Matheus (2025). Kworkflow. Archived in Software Heritage: swh:1:dir:90dc41328e09271597eb1f4f47d8a4c2e972a5bb.

Steinmacher, I., Wiese, I., Conte, T. U., and Gerosa, M. A. (2015). Increasing the self-efficacy of newcomers to open source software projects. In *2015 29th Brazilian Symposium on Software Engineering*, pages 160–169.

Tan, X., Chen, Y., Wu, H., Zhou, M., and Zhang, L. (2023). Is it enough to recommend tasks to newcomers? understanding mentoring on good first issues. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 653–664.

Xiao, W., He, H., Xu, W., Tan, X., Dong, J., and Zhou, M. (2022). Recommending good first issues in github oss projects. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1830–1842, New York, NY, USA. Association for Computing Machinery.