Guidelines for Boosting Long-Lasting FLOSS Contributors

David Tadokoro, Rafael Passos, Paulo Meirelles

Guidelines for Boosting Long-Lasting FLOSS Contributors

David Tadokoro
University of São Paulo, Brazil
davidbtadokoro@usp.br

Rafael Passos
University of São Paulo, Brazil
rafael@rcpassos.me

Paulo Meirelles
University of São Paulo, Brazil
paulormm@ime.usp.br

***Abstract*—FLOSS (Free/Libre and Open Source Software) development is a validated approach to producing cutting-edge software solutions used by governments, companies, and society at large. At the core of all FLOSS projects are communities composed primarily of developers who evolve and maintain the software while devising rules for the development process. A common problem is the renewal and/or scaling of the workforce, which is not about encouraging waves of new and sporadic contributors but about fostering long-lasting ones that can have a more profound impact on the project. This work presents guidelines for mentoring trainees to build skills and gain experience essential to becoming valuable assets in most FLOSS ecosystems. Beyond a reasonable software development background, previous experience in FLOSS development is not a requirement. These guidelines are a product of a four-month university course where students had to (1) learn the fundamentals of Linux kernel development and send patches to a subsystem, (2) contribute to supporting tools to the GNU/Linux ecosystem, (3) experience software packaging in the context of Debian, and finally (4) contribute to a chosen FLOSS project. The method reproduces the natural way a "self-taught" contributor would enter a FLOSS ecosystem (heavily inspired by how Debian does it), though in a more focused and immersive environment. Through on-site workshops, accessible and knowledgeable mentors, and close and constant monitoring, we simulated a smaller FLOSS community where trainees (inexperienced contributors) learned from mentors (maintainers); we argue that this simulated community can be more efficient than a real one as feedback is faster and more adapted to each contributor, and concepts can be abstracted and simplified for easier absorption. Our results show that the method fostered (1) developers who are more confident in contributing to any FLOSS project and going beyond the one-time contributions and (2) the enhancement of fundamental hard skills (like Git and Web/email-based contribution models) and soft skills (like communication and feedback assimilation) for any FLOSS project. Even though deep technical knowledge becomes mandatory for every contributor in a specific project, we claim that implementing the proposed guidelines can quickly nurture tens of developers with a solid base for becoming long-lasting contributors to FLOSS projects.**

I. Introduction

FLOSS (Free/Libre and Open Source Software)[1] projects, more often than not, rely on a community of developers composed of people who contribute to the project by submitting code changes, reporting/testing bugs, and the like, called *contributors* and developers who maintain the project by reviewing and integrating code changes, defining the development direction, and so on, called *maintainers*. In this sense, every FLOSS project sustainability depends on the renewal of its workforce through new contributors that arrive in the project, known as *newcomers*.

Extensive work has been done to investigate the entry barrier of newcomers into the FLOSS ecosystem. Steinmacher et al. [1], using a model of entry barriers, inspected how to increase the perceived self-efficacy of newcomers in their first contributions, covering the motivation and confidence aspect of newcomers. In two studies, Balali et al. [2], [3] analyze mentoring as an educational approach to introduce newcomers to FLOSS, focusing on mentors' and mentees' pain points and challenges while presenting strategies to alleviate them. More recent research works [4], [5] look to how recommended tasks for newcomers, usually called *Good First Issues*, can promote new contributors and strategies for doing so.

Nonetheless, these works remain at the stage of *one-time contributors*—those who make only a single, usually low-impact contribution. Such contributions are often simple and demand more time from maintainers to guide newcomers than they offer in return. Meaningful impact arises only when newcomers continue contributing and move beyond one-time contributions. Furthermore, while valuable for improving newcomer onboarding, these studies address only specific, isolated aspects of the FLOSS contributor journey.

To help filling the described gap, this work presents guidelines that can be used to train developers to become long-lasting contributors to most FLOSS projects by equipping them with the necessary skills and experience. The guidelines were designed for a multi-month mentorship program that emerged from a university course focused on training students in FLOSS development with strong ties to the Linux kernel project and its ecosystem.

With close support from multiple mentors, regular on-site workshops, rich teaching materials, and lightweight monitoring mechanisms, we simulate a FLOSS community centered on contributors capable of quickly grooming many developers into solid assets for FLOSS. In this respect, the guidelines offer a novel approach to boosting long-lasting FLOSS contributors.

The remainder of the text is structured as follows: Section II describes the training experience of the university course and the methods used to collect and analyze data; Section III displays the results obtained and discusses how they inform us regarding the effectiveness of the guidelines and its limitations; Section IV presents the training guidelines succinctly and

[1]https://gnu.org/philosophy/floss-and-foss.html

directly for easy reference; and Section V concludes this work with opportunities for future research.

II. Materials and Methods

This section describes the resources utilized in the training experience, the approach used, and the methods employed to collect and analyze data.

A. Resources Utilized

Physical and human resources, teaching materials, and monitoring mechanisms were required to conduct the training. Traditional talks and on-site *workshops* were central, necessitating a weekly venue with a capacity of thirty participants, equipped with tables and high-bandwidth Internet, that supported both lectures and group activities. There were three *mentors* and a group of 27 *trainee developers*. Teaching *materials*, such as presentations, tutorials, and reference materials, were produced or provided by mentors throughout the training. Lastly, simple mechanisms to *monitor progress* were implemented, such as constant logging of activities and quick presentations from trainees.

B. Training Approach

We implemented the training in a four-month university course involving 21 undergraduate and 6 graduate students from diverse academic backgrounds. The description of the training approach is presented independently of this specific context as much as possible to increase its reproducibility. Our approach aimed to immerse trainees in diverse FLOSS development models and communities. The program used the Linux kernel and Debian projects as case studies and was structured into four phases:

a) Phase 1 – Linux kernel development: The trainees began by doing tutorials[2] to learn the fundamentals of Linux development, like environment setup, developer workflows, and device driver anatomy, all in the context of a specific subsystem. This phase concluded with guided patch submissions, where trainees first sent patches to mentors for validation before sending them as real contributions to Linux.

b) Phase 2 – Linux supporting project: After experiencing Linux development, the trainees moved from the email-based model to a web-based project that supports the Linux ecosystem; for that, we chose the *kworkflow* tool [6].

c) Phase 3 – Debian packaging: While keeping in the Linux ecosystem, the trainees moved to Debian packaging, a layer more palpable and closer to users than the Linux kernel and supporting tools.

d) Phase 4 – Independent FLOSS Contributions: Finally, the trainees selected FLOSS projects and contributed independently.

Classes, expository sessions, and workshops were held throughout the training on-site. Our approach simulated real community dynamics, with mentors acting as maintainers who guided and instructed trainees and occasionally reviewed their contributions. Close mentor-trainee interaction enabled efficient and adaptive feedback. During the first three phases, contribution suggestions were carefully curated to ensure they were accessible to beginners yet substantive, avoiding trivial tasks to foster meaningful learning.

[2]https://flusp.ime.usp.br/kernel

C. Data Collection and Analysis

To evaluate the effectiveness of the training approach, we gathered qualitative data from three sources:

a) Trainees Blog Posts: The trainees documented their progress throughout the phases and related activities via blog posts, simulating developer board logs.

b) End-of-Training Survey: A 47-question survey collected trainees' demographics, perceptions of learning outcomes, and course structure feedback. Questions used Likert scales, yes/no, and open formats.

c) Mentor Interviews: Interviews captured mentors' observations about the training process and trainees' development.

Survey data was aggregated and categorized (positive, neutral, negative) for key questions. Blog posts and interviews were analyzed qualitatively to identify recurring patterns and insights.

III. Results and Discussion

This section shows the data analysis core results and discusses how they corroborate the training guidelines effectiveness.

A. Key Findings

The training approach successfully shifted the trainees' perceptions and built foundational skills for contributing to FLOSS projects despite many trainees having no prior contribution experience. Some highlights that showcase this are:

a) Changing Perceptions and Confidence: While most trainees were familiar with the concept of FLOSS, only a quarter had contributed before. By the end of the training, around 75% reported feeling more capable and confident to contribute to any FLOSS project, showing a clear sign of evolution.

b) Technical Growth: Initial struggles with command-line tools and Linux environments were common but were gradually overcome. 90% of trainees felt their Git proficiency improved, and 85% recognized it as an essential skill for any FLOSS contributor. Hands-on experience with both email and web-based workflows made trainees more aware and familiar with the core concepts of FLOSS development.

c) Mentorship and Workshops: Mentorship and on-site workshops were pivotal. Over 85% of trainees cited these as critical for their learning and evolution as FLOSS contributors. With the close support of mentors, many transitioned from dependency to autonomy during the training experience, managing to contribute to real projects and accomplish the remaining tasks with minimal guidance by the end.

All trainees made (at least) four contributions to four different FLOSS projects within a relatively short period, providing them with a broad overview of FLOSS development models. Rather than simply repeating the role of a one-time contributor, trainees were required to refine both their hard and soft contributor skills to produce meaningful contributions and complete the training activities.

Starting with Linux development was certainly a shock for most trainees; nevertheless, it proved beneficial in the long term, as it encouraged independent problem-solving and deepened their understanding of core FLOSS workflows and dynamics.

Well-designed teaching materials were essential to the success of the training, as, on the one hand, many of the hurdles and blockers reported were attributed to issues in the tutorials, while, on the other hand, most of the knowledge was conveyed to the trainees through these materials.

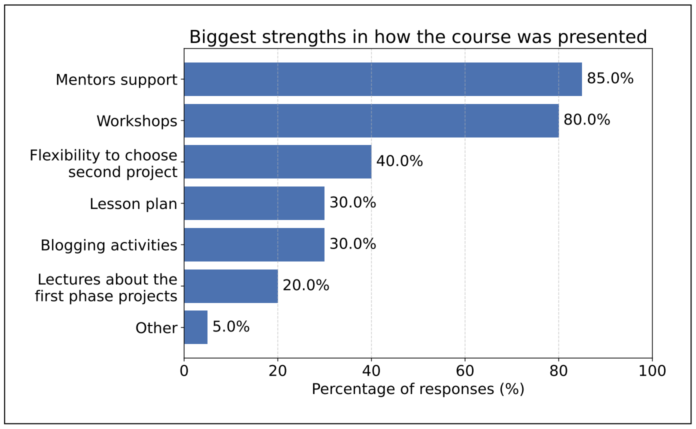

Fig. 1. Bar plot of the course strengths according to trainees.

Figure 1 gives an overview of the university course strengths according to the trainees' perspective, enforcing the importance of close mentoring and workshop sessions.

B. *Limitations*

Some limitations of the study methods may affect their validity and applicability. The student-professor dynamic of a university course, which is tied to grades, can introduce bias in the trainees' level of dedication. Students may feel motivated to invest more effort but may do only the minimum required to complete the course. Nonetheless, motivation is a key factor in becoming a long-term FLOSS contributor and this work does not aim to analyze that aspect in depth.

Conversely, the guidelines are closely tied to the Linux ecosystem. Even though the results suggest that this case study choice produced positive outcomes, a different set of FLOSS projects might prove more effective or lead to different (yet still valuable) outcomes. From the authors' perspective, a variation of the guidelines focused solely on the Debian project, exploring its diverse development contexts thoroughly, should be considered.

Finally, although this work focuses on common characteristics of FLOSS projects, each project has its technical particularities, including specific technologies and domain knowledge. These aspects must be addressed at some point in a contributor's journey into a particular project, but the guidelines do not cover them.

IV. Guidelines for Boosting Long-Lasting Contributors

This section presents practical guidelines to mentor and train individuals into confident, autonomous, and enduring contributors to FLOSS projects, based on the methodology used in a four-month university course.

A. *Resources Required*

Implementing the guidelines relies on a minimal but deliberate set of physical, human, and informational resources. These are summarized in Table I.

TABLE I
Resources Required for Training Long-Lasting FLOSS Contributors

Resource Type	**Description**
Physical Space	A weekly-available room with capacity for twenty to thirty people, tables for group work, and high-bandwidth Internet access.
Mentors	Personnel experienced in FLOSS to guide and support trainees. Mentors should also be capable of reviewing contributions. Ideally, one mentor per ten trainees.
Trainees	Developers with basic programming experience and Git knowledge; prior FLOSS experience is not required.
Teaching Materials	Tutorials, presentations, and reference documentation provided and maintained by mentors.
Monitoring Tools	Lightweight mechanisms such as weekly deliverables and short in-person presentations to track progress.

B. *Training Structure and Phases*

The training follows a progressive, immersive structure that exposes trainees to different FLOSS development ecosystems and models. To support this progression, it is divided into sequential phases, though some overlap may occur (e.g., continuing a contribution from a previous phase). Each phase develops technical (hard) and interpersonal (soft) skills, gradually strengthening autonomy and community engagement.

We do not prescribe a fixed duration, but we recommend targeting programs lasting three months to one academic semester. This time frame allows concepts to be absorbed and experiences to develop without overloading the trainees while still being short enough to maintain engagement. In this respect, we suggest the following phases structure and approximate durations.

a) Initial Phase (two months): This phase fosters discipline and autonomy while introducing the trainees to key FLOSS concepts, such as the nature of contributions, how to submit them, and how to engage in the review process. We recommend using an email-based development model, like that of the Linux kernel or Git to familiarize participants with this less intuitive workflow. This model exposes the trainees to real-world contributor practices: generating and sending patches, receiving email feedback, interpreting inline reviews, and more. If opting for the Linux kernel, we suggest focusing on a single subsystem with an active, welcoming community and, ideally, a veteran willing to support the program. The phase should conclude with trainees submitting contributions first to mentors for simulated review and feedback, followed by official submission to maintainers and the broader community.

b) Independent Contributions Phase (two months): Following the structured initial phase, the trainees should apply their skills independently to a different FLOSS project. By "independent", we mean without mentors curating tasks, reviewing contributions in advance, or selecting the project. However, mentors should continue offering general support through teaching materials, workshops, and availability for guidance. While projects need not be predefined, mentors should validate the suitability of selected projects. Preparing a list of recommendations is encouraged. The only requirement is that projects must be web-based (e.g., hosted on GitHub or GitLab).

c) Complementary Short Phase (two weeks): A brief experience (ideally self-contained in two workshop sessions) can further enrich trainees' understanding of FLOSS. This phase should focus on a single project or community, offering a deep dive without heavy technical demands. We strongly recommend Debian packaging for this purpose, as it allows the trainees to perform straightforward tasks while gaining a broad view of the ecosystem with minimal overhead.

We recommend implementing the program through weekly four-hour sessions, with a twenty-minute break after the first two hours. While these hours can be split across two days, our experience shows that a single session maintains trainee engagement and avoids context-switching overhead. Sessions should occur on-site and include workshops, lectures, and other teaching dynamics as needed by mentors. Naturally, tasks will extend beyond these sessions, as trainees must study materials and plan, produce, and revise their contributions.

C. Monitoring Recommendations

To ensure consistent progress while promoting autonomy, mentors should implement monitoring mechanisms tailored to the specific environment. Nevertheless, we present the following recommendations:

- **Self-Progress Logging:** Trainees maintain blogs or logs documenting their work, reflections, and blockers.
- **Initial Phase Reviews:** All contributions in the initial phase should be reviewed internally before public submission.
- **Periodic Check-ins:** Using a short seven-minute pitch format, trainees give monthly presentations on their progress.
- **Soft Tracking in Workshops:** During on-site workshops, mentors unobtrusively monitor trainees to identify who is more advanced and who may be struggling.

D. Important Notes

Several aspects are critical for successfully boosting long-lasting contributors:

- **Challenge Early, Not Late:** Starting with a technically demanding project such as the Linux kernel, although intimidating, accelerates learning and builds resilience.
- **Prioritize Mentorship:** Close, in-person mentorship is pivotal for simulating a high-feedback FLOSS environment, which accelerates learning compared to typical asynchronous online communities.
- **Curate Tasks:** Contributions must be carefully selected, challenging enough to stimulate growth but accessible enough for beginners to engage meaningfully.
- **Focus on Core Skills:** Emphasize broadly applicable skills such as Git proficiency, email and web-based collaboration, communication, and community etiquette.
- **Simulate the Community:** Workshops and mentorship should recreate the interpersonal dynamics of FLOSS communities while enhancing clarity and feedback.

V. Conclusion

Acknowledging the lack of investigation into methods for boosting long-lasting FLOSS contributors beyond one-time contributions, this study introduces a bold training approach to address this gap. The approach is presented as a set of guidelines and emerged from a four-month university experience. Results indicate that implementing these guidelines can (1) increase trainees' confidence in contributing to any FLOSS project and (2) enhance both hard and soft skills that are fundamental and widely applicable across most, if not all, FLOSS projects. The analysis of results also highlighted the importance of close support from mentors and physical proximity to both peers and mentors through on-site workshops, which simulate a FLOSS community, although it is one more centered on the contributor's needs.

Future work involves testing guideline variations, particularly with different sets of FLOSS projects. A version focused solely on the Debian project is especially promising given its significance in the FLOSS ecosystem, detailed workflow documentation, and a community that is open and welcoming to newcomers. Additionally, repeating the experience while expanding survey data could provide deeper insights for refining the guidelines and better assessing their impact and limitations. Key data points include: trainees' self-assessment of hard skills (e.g., Git and CLI tools) before and after training; trainee demographics (e.g., age and language proficiency); and whether and how trainees continued contributing to FLOSS, while taking care to avoid panel bias, since those who continue may be more likely to respond.

Reproducibility and Data Availability

The study presented here is based on a work in progress, so we will only make available the subset of the data collected necessary for reproducing the results presented in Section III. The subset and other artifacts are stored in this Zenodo repository[3].

[3]https://doi.org/10.5281/zenodo.15699908

Acknowledgment

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 and the São Paulo State Data Analysis System Foundation - SEADE (Proc. 2023/18026-8), Brazil.

References

[1] I. Steinmacher, I. Wiese, T. U. Conte, and M. A. Gerosa, "Increasing the self-efficacy of newcomers to open source software projects," in *2015 29th Brazilian Symposium on Software Engineering*, 2015, pp. 160–169.

[2] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa, "Newcomers' barriers. . . is that all? an analysis of mentors' and newcomers' barriers in oss projects," *Comput. Supported Coop. Work*, vol. 27, no. 3–6, p. 679–714, Dec. 2018. [Online]. Available: https://doi.org/10.1007/s10606-018-9310-8

[3] S. Balali, U. Annamalai, H. S. Padala, B. Trinkenreich, M. A. Gerosa, I. Steinmacher, and A. Sarma, "Recommending tasks to newcomers in oss projects: How do mentors handle it?" in *Proceedings of the 16th International Symposium on Open Collaboration*, ser. OpenSym '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3412569.3412571

[4] X. Tan, Y. Chen, H. Wu, M. Zhou, and L. Zhang, "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)*, 2023, pp. 653–664.

[5] W. Xiao, H. He, W. Xu, X. Tan, J. Dong, and M. Zhou, "Recommending good first issues in github oss projects," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1830–1842. [Online]. Available: https://doi.org/10.1145/3510003.3510196

[6] Siqueira, Rodrigo and Tavares, Matheus, "Kworkflow," 2025, archived in Software Heritage: swh:1:dir:90dc41328e09271597eb1f4f47d8a4c2e972a5bb.