Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Bárbara de Castro Fernandes

How to contribute to the Linux kernel project: the case of the FLUSP group

São Paulo
December 2019

How to contribute to the Linux kernel project: the case of the FLUSP group

MAC0499 – Supervised Capstone project.

Advisor: Paulo Meirelles
Co-advisor: Melissa Wen

São Paulo
December 2019

Resumo

Comunidades de software livre dependem de um fluxo contínuo de novos contribuidores para continuarem operando. Devido aos laços fracos que unem os participantes à comunidade escolhida, a taxa de evasão de seus membros é alta, e esses tendem a desistir do projeto em face às primeiras adversidades. Como resultado, diversos projetos fracassaram em virtude da participação insuficiente de voluntários. Este trabalho tem como objetivo analisar o papel que grupos de estudo desempenham na subsistência dessas comunidades através do estudo de caso do FLUSP, um grupo de extensão do Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP). Analisamos as contribuições enviadas por seus membros para um subsistema do *kernel* Linux, o Industrial Input/Output (IIO), através de coleta de dados dos *patches* enviados e da aplicação de um questionário a cinco de seus membros mais ativos.

Palavras-chave: Software Livre, Engenharia de Software Empírica, Mineração de Repositórios de Software.

Abstract

Free/Libre/Open Source Software (FLOSS) communities depend on a continuous stream of contributors to keep ongoing. Due to the weak links connecting participants to the chosen community, the rate of dropouts is high, and its members tend to give up in the face of the first adversities. As a result, many projects have failed due to insufficient volunteer participation. This work aims to analyze the role that study groups have in the subsistence of those communities through the case study of FLUSP, an extension group from the Institute of Mathematics and Statistics of the University of São Paulo (IME-USP). We analyzed the contributions sent by its members to a kernel Linux subsystem, the Industrial Input/Output (IIO), through data gathering of the patches sent and through the application of a questionnaire to five of its most active members.

Keywords: FLOSS, Free Software, Open Source Software, Empirical Software Engineering, Mining Software Repositories.

Contents

List of Abbreviations

CCSL	FLOSS Competence Center
FLOSS	Free/Libre Open-Source Software
FLUSP	FLOSS at USP
GSoC	Google Summer of Code
IIO	Industrial Input/Output
IME	Institute of Mathematics and Statistics
KDD	KernelDevDay
UFBA	Federal University of Bahia
USP	University of São Paulo

List of Figures

Chapter 1

Introduction

Free/Libre/Open Source Software (FLOSS) is a term used to encompass all software released under a license that allows its users the freedom to inspect, use, modify and redistribute the source code (Crowston *et al.*, 2012). Since its inception by the Free Software (in the 1980s) and the Open-source (in the 1990s) movements, the FLOSS usage has grown both in the industry and for personal use. Its popularization also increased the interest of understanding its practices of development and community organization. However, despite many investigations about it, FLOSS remains a commonly misunderstood phenomenon, with significant gaps between theory and practice over more than two decades (Osterlie and Jaccheri, 2007).

In the world of software engineering, one of the research opportunities which arise from FLOSS studies is the motivations that drive individuals to participate in these projects and what prompts them to continue contributing (Scacchi, 2010). Much of this is because FLOSS projects are commonly community-based, and the community depends on a continuous stream of contributors to keep going. Due to the weak links connecting participants to the chosen group, the rate of dropouts is high, and its members tend to give up in the face of the first adversities (Silva, 2019). As a result, many projects may fail due to the insufficient number of contributors.

One way of helping newcomers in those communities is through their involvement in study groups. The assistance offered by study groups can decrease the learning curve associated with entering a FLOSS project and also keeping the individual involved. Notwithstanding many advantages for academic teaching and professional training, FLOSS development groups are not commonly found in Brazilian universities.

With this concern, this work looks into the role played by a university group for engaging students in FLOSS development and improving the member's retention of these communities. We examined the case of FLUSP (FLOSS at USP)[1], a group of graduate and undergraduate students at the University of São Paulo (USP) that aims to contribute to FLOSS projects. Its members have sent contributions to several projects, such as to the Linux kernel, Git, and GCC, as well as having participated in the Google Summer of Code initiative and taking part in FLOSS conferences.

To conduct this investigation, we applied qualitative research methods, engaging in participant and non-participant observation of its members, two study methods often used in social research. We paid closer attention to the contributions of its seven most active members, in addition to administering them a questionnaire related to their participation in the group. We studied the methods of engagement used by this group, such as pair programming, internal reviewing, and blog posting, and analyzed their efficacy. We also examined

[1] https://flusp.ime.usp.br

the patches sent to FLUSP's internal mailing list, keeping track of which ones were sent to the community's official list and, among those, the ones who were accepted and merged into the project's source code. We categorized them according to the author, contribution type, and whether they were co-developed or not.

Besides that, we also began participating in its activities from January 2019 to November of the same year. We took part in FLUSP's weekly occurrences during this period, which included event planning, bureaucratic issues, and participation in two conferences. Immersion on the development process of FLOSS software projects happened through the sending of contributions as a community developer. As a consequence, we sent ten patches through the course of those ten months to one of its most active projects, the subsystem of the Linux kernel known as Industrial Input/Output (IIO)[2]. The IIO subsystem was created in 2009 by Jonathan Cameron and aggregates drivers to devices that perform analog to digital and digital to analog conversions, such as pressure sensors, temperature sensors, accelerometers, and magnetometers. We decided to select it as the focus of this study since it received the most contributions from the FLUSP group.

We have also sent a questionnaire to five of the seven studied subjects related to their experience at FLUSP and regarding the FLOSS projects to which they contribute. One of the participants did not answer the questionnaire due to having participated in the elaboration of the questions. Afterward, we analyzed the gathered data, inferring whether the group is fulfilling its role of bringing newcomers to the FLOSS community and stimulating their permanence there. We used the results to investigate whether this group is indeed fulfilling its role of bringing contributors to those communities and increasing their rates of permanence. The questions they had to answer can be found in Appendix A.

Our results indicated that FLUSP fulfilled its role of helping the introduction of its members in the FLOSS community. Its philosophy of knowledge sharing, internal review, and pair programming served to mitigate the initial difficulties new contributors often find when they begin to navigate a FLOSS project.

1.1 Research Design

In this work, we used qualitative research strategies for data collection and interpretation: the case study and participant and non-participant observation. Qualitative methods are research techniques for collecting and analyzing data focused on the comprehension of the reasons and motivations behind a certain occurrence (Seaman, 1999). To better understand the dynamics involved in the kind of voluntary contribution that takes place in FLOSS projects, we chose the FLUSP group to be the subject of our case study.

A case study is a qualitative research method that aims to provide the necessary tools for a researcher to gain an understanding of complex events such as the roles and rules involved in the dynamics of a specific community through investigating its particular instances. It is with the help of the evidence obtained through a case study that models developed for a theory can be tested to see if they correspond to reality.

We methodologically study the participation of six of FLUSP members to the IIO subsystem, in addition to having also contributed to it ourselves through the sending of 10 patches. The importance of the participant observation can be seen in the gathered knowledge the researcher experiences through immersion in the phenomenon at hand and is thus able to gain further insight. In the same way, it is also valuable to engage in non-participant observation to try to understand the parties' actions without external interference.

[2] https://01.org/linuxgraphics/gfx-docs/drm/driver-api/iio/index.html

The FLUSP mailing list received a total of 221 patches between 4th October 2018 and 25th October 2019. The participants were chosen based on their frequency of contribution and permanence in the group. To make the data analysis possible, we subscribed to the FLUSP internal mailing list and inspected all the patches sent through it since its inception. After that, we crossed the patches against those sent to the IIO mailing list. We organized the data gathered in a spreadsheet arranged by title; date sent; main author; whether it was co-developed or not; co-authors (if any); patch type; if it was sent to IIO; if it was accepted in IIO; and links to the patch in FLUSP and IIO mailing lists.

When two or more developers write a patch, it receives in its description a tag known as "co-developed by". We separated the patches between those that were co-developed and not to study the frequency they are done and what drives contributors to create patches together.

We observed that between different versions of a patchset, the number of patches in it could vary. To investigate the number of unique patches sent by each programmer, we considered the number of patches of the final version. The final version of a patch set is the one that was sent to the IIO mailing list, independently of it being accepted and merged to the maintainer's repository or not.

1.2 Organization of the capstone project

In Chapter 2, we present a conceptualization of the terms that are used throughout this work, in addition to furnishing a background on FLOSS history and, especially, of the Linux kernel. We also report the main events that happened in FLUSP's short existence and a brief account of USP's involvement in the FLOSS world.

In Chapter 3, we probe more thoroughly into FLUSP's internal structure, describing the tools of which the group make use and how they are used in the training process of its members. Besides that, we provide a brief description of the participants involved in the study, along with exposing the results we observed in our research. This includes the examination of the patches sent by the group's members, as well as an analysis of the answers given by the studied subjects to the questionnaire we created.

Finally, in Chapter 4, we detail the conclusions we were able to observe through our research and give motivation to the production of further works on the subject.

Chapter 2

Background

In this chapter, we introduce a brief account of the Linux kernel history and its organization, and discuss the GNU Project and the concept of FLOSS, besides describing how the Linux code is organized and how the contribution flow to its subsystems work. Also, we present the history of GNU/Linux operating system in the University of São Paulo and the creation of the FLOSS at USP group.

2.1 The Linux Kernel

Linux is an operating system kernel created by Linus Torvalds, a software engineer from the University of Helsinki. Initially designed for i386 machines and as a hobby[1], it found inspiration in MINIX (Tanenbaum and Woodhull, 2011), an operational system created by Andrew Tanenbaum for educational purposes in 1987.

There are two main kinds of kernels, the microkernel and the monolithic kernel. In the microkernel, this part of the operating system has the minimum amount of software necessary to make it runnable. Besides, the user services and kernel services are kept in different address spaces. The MINIX is an example of such kind of kernel.

Also known as "The Big Mess" (Tanenbaum and Woodhull, 2011), in the monolithic kernel the entire operating system works in kernel space. And it is not uncommon to have tasks like device drivers, file system and memory management being performed by it. We can cite Linux as a kind of monolithic kernel. The duality of monolithic kernel and microkernel gave rise to many feuds, including one between Tanenbaum and Torvalds. Details about this discussion can be seen in the Usenet thread created by Tanenbaum in comp.os.minix on January, 29th, 1992 (DiBona and Ockman, 1999).

Soon Torvalds' project grew with the help of the GNU Project and was released under the GNU General Public License as the GNU/Linux operating system. Richard Stallman started the GNU Project in 1983 to replace all the most critical software in the computer by a FLOSS alternative. In the vision of Stallman, every software should grant its users four freedoms: the freedom to use, study, share and improve it[2]. Those freedoms form the basis of the free software movement, and the GNU General Public License exists to guarantee all software released under this license have them respected.

Since the source code of the Linux kernel was first released to the world in 1991, GNU/Linux has grown to become one of the most famous operating systems that ever existed. It is now considered the most successful example of a FLOSS project. Nowadays,

[1] https://groups.google.com/forum/#!topic/comp.os.minix/dlNtH7RRrGA
[2] https://fsfe.org/freesoftware/basics/4freedoms.en.html

all the supercomputers in the TOP500 list use it as their operational system[3].

The GNU/Linux operating system can be divided into four main layers: the hardware, the kernel, the shell and the applications. The hardware consists of all the physical components of the computer. The kernel is the core component of any operational system, providing communication with the hardware and handling the system tasks. The shell operates as an interface that takes commands directly from the user and executes kernel commands. The applications are the programs that supply most of the functions of the computer. In this work, we focus on the kernel layer.

2.1.1 The Linux Development

Some initiatives both from industry and academia have studied FLOSS development characteristics (Fitzgerald, 2006; Raymond, 1999). One of the first essays to broach this subject was Eric S. Raymond's The Cathedral and the Bazaar. In this seminal work, Raymond exposed the experience he gained by managing a project of this kind, the *fetchmail*.

Raymond exemplified the bazaar model by the style of development employed by the Linux project. This model is characterized as bordering on the "promiscuous" side. It is based on early and often releases and delegation of everything possible. This goes directly against the Cathedral style of development, that perpetuates that the final user must see the least amount of bugs as possible in the code. The bazaar style rationale is exemplified by a claim the author called Linus's Law: "given enough eyeballs; all bugs are shallow."

The Linux kernel functions can be divided into the following five parts (Corbet *et al.*, 2005):

- Process management: a computer process is a program in execution. The process management of an operating system concerns process creation and termination, as well as the scheduling algorithms employed to share the CPU usage;
- Memory management: most computers follow a memory hierarchy, having a small amount of fast, expensive, volatile memory (Random Access Memory, or RAM), and a larger quantity of slow, cheaper, non-volatile memory (Read-Only Memory, or ROM). Memory management pertains the part of the operating system that deals with this hierarchy, in addition to memory allocation and deallocation;
- Filesystems: the filesystem is involved in information storage and retrieval. Linux follows the Unix methodology that asserts that almost anything can be treated as a file;
- Device control: a device driver is the part of an operating system code involved in communication with the hardware. A driver provides device-specific code to perform input and output operations;
- Networking: part of the operating system that controls the delivery and collection of packets through a communication network. It also performs packet routing and addresses translation.

Since the kernel has long stopped being a project able to be managed by a single person, it now works with a chain of trust model. Most subsystems have a maintainer, who is the person responsible for the patches that go through its repository. Linux subsystems are all

[3] https://www.top500.org/

managed by volunteers, who have the duty of ensuring the code quality. Each maintainer has its version of the kernel source tree and uses version-control tools, such as Git, to trace information about the patches contained on it.

All the subsystems have a staging directory, where the code for its drivers and filesystems stay until they are well developed enough to become part of the kernel source tree. Greg Kroah-Hartman currently maintains the staging tree.

Periodically, Torvalds' source tree begins accepting pull requests. When that happens, the maintainers ask Torvalds to pull the patches they have selected from their version of the source tree, thus becoming part of the mainline branch.

One thing that facilitates the merging of their work is the kernel modularized structure. Modules make it possible to extend the functionalities provided by the kernel in runtime. They render more flexibility to the system, allowing the dynamic linking and unlinking of pieces of code while the Linux is running.

Most of the code development done in Linux is made through mailing lists. In this work, our focus is on the Industrial Input/Output (IIO) subsystem, which is part of the device control portion of the kernel. The IIO aims to provide support for devices that perform either an analog-to-digital conversion, digital-to-analog, or both. It is currently maintained by Jonathan Cameron, from Huawei, and has been since its inception in 2009.

Cameron started working on an academic project called SESAME, in which athletes were equipped with movement sensors on their bodies. There were not many platforms out there to do this. He asked on the Linux kernel mailing list and concluded that he would have to make something new. That is how he started planning the IIO subsystem.

2.1.2 The Contribution Workflow

To begin contributing, a contributor must subscribe to the subsystem mailing list, besides making a copy of the subsystem source code. It is also necessary to set the development environment and to create a virtual machine to test the changes made to the code without crashing the whole operational system the user is using in the code development.

The entire contribution process begins by finding an issue in which to work. This process can be done by accompanying the subsystem mailing list, looking for the issues page in the project Wikipedia, or asking directly to a maintainer.

Once an issue has been selected, the contributor must locate the sections of the code that must be modified or where a new code must be added to fix the problem at hand. Each patch must be composed of small pieces of code and must address one issue at a time. It is common practice to break a series of modifications in what is called a patch set. A patch set is nothing more than a grouping of modifications, all related to the same problem.

After the code has been modified, the changes must be tested. The testing part consists of compiling the source code to see if it breaks the code and running the existing test files related to the sections of code modified in the patch.

Through the help of a contribution group, the created patch can be sent to an internal mailing list. In this mailing list, the modifications are again tested by other contributors to find issues the patch developer might have potentially missed. After those issues are dealt with by the developer, the patch is again sent to the internal mailing list until no other issues are found. When this happens, the patch can be sent to the official mailing list.

```
<subsystem>: <what changed>
<blank line>
<why this change was made>
<blank line>
<footer>
```

Listing 2.1: *The format a commit message should have*

The sending of patches is managed with the help of another of Torvalds' creation, the version-control tool named Git. In Git, each patch roughly corresponds to one commit. Every commit must have a message describing what the patch does. The first line of the message is the title, which briefly describes what the proposed modifications do, and the last lines contain the footer. It is in the footer that information about the people directly involved in its development is placed. The format a commit message can be seen in Listing 2.1[4].

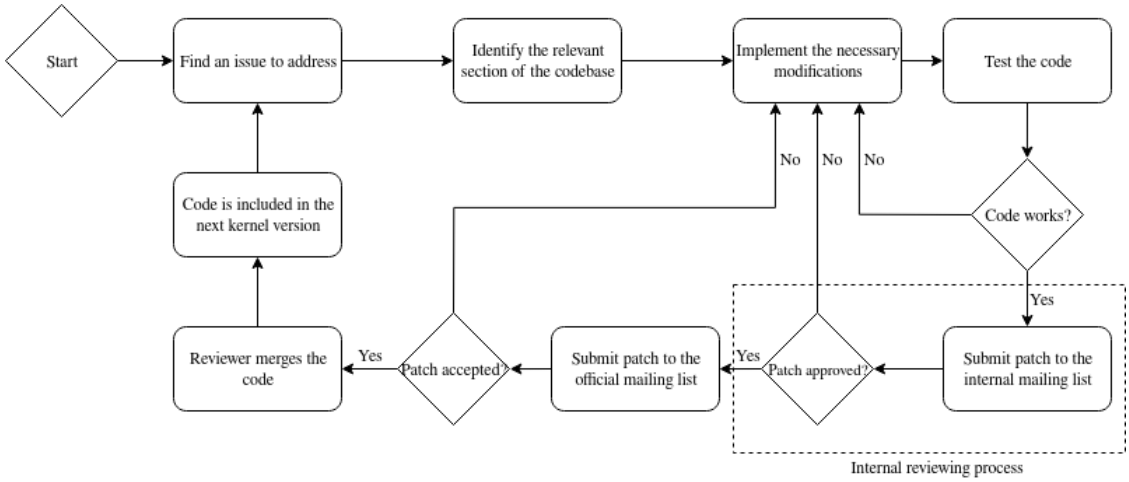


Figure 2.1: *Contribution workflow in a Linux subsystem*

After the code is sent to the official mailing list, one of the maintainers is in charge of testing the code. If any issue is found on the code, the developer must apply the suggested modifications to have their patch accepted. If no issue is found, the maintainer accepts the patch and have its code merged to the kernel source tree version of their subsystem. In this way, the patch is added to the next version of the kernel released. We illustrate the whole contribution process in Figure 2.1.

2.2 FLOSS at USP Group

The Institute of Mathematics and Statistics of the University of São Paulo (IME-USP) has a history with FLOSS since the installation of the GNU/Linux operational system by Prof. Marco Dimas Gubitoso, in 1993. Today students at the institute can benefit from the usage of the "Rede Linux ", created in 1998 to provide them with the necessary computational resources. The university involvement with FLOSS does not end there, though. In 2006, FAPESP and the rectory released money for the construction of the FLOSS Competence Center (CCSL). Two years afterwards, its statute was written, and this legally marked the creation of the CCSL.

In this scenario, FLUSP acted as an empowerment tool that USP students could use to contribute to complex FLOSS projects, such as the Linux kernel. The idea of creating FLUSP raised during the subject of Extreme Programming Lab. Also known as LabXP, this discipline aims to provide students with an opportunity to engage in a software project and make use of the Agile development methodology. Four of the LabXP students went on to establish FLUSP.

[4] https://docs.flatcar-linux.org/container-linux-config-transpiler/contributing/

During the subject, a Computer Science master student from IME-USP, strived to transmit all the knowledge he had about FLOSS, the Agile methodology and the Linux kernel. It only took a month for the students to send their first patches, and by the second month, they did not need his help anymore. Prompted by him, they started writing blog posts, since this was a huge differential for the ones who wanted to apply to GSoC.

In December of 2018 began the race for the members who applied to this program. As a result, three FLUSP members had their proposals accepted and went on to contribute to GCC, the IIO subsystem and Git. It was also in this month, on the 10th, that FLUSP officialization as an extension group took place.

FLUSP currently has around six active members. It has also study groups dedicated to Debian, GCC, Git and the Linux kernel. It holds weekly meetings in which its members gather to work on these projects.

Each of the group projects is conducted as independent entities. The participants have the prerogative concerning how the subgroup is managed. They also define how and when its meetings are take place.

FLUSP has three main communication channels: a mailing list, an IRC channel and a Telegram group. There is also a bot that performs integration between its IRC channel and Telegram group, allowing the messages sent in one of them to be available in the other.

FLUSP has an internal revision system in which members of the group review each other patches before sending them to a FLOSS project (i.e., IIO mailing list). This approach helps avoiding the sending of patches with minor mistakes and it also teaches its members to review patches and become more acquainted with the subsystem code.

Moreover, in its short story, FLUSP has already organized several FLOSS events: the install fest 2019 at IME-USP, a Git workshop, and a marathon of collaborations to the kernel IIO subsystem (KernelDevDay). FLUSP also supports part of linuxdev-br 2019 edition. In particular, the KernelDevDay (KDD) gathered 21 people and 16 patches were sent to the IIO mailing list.

Chapter 3

Discussion and Results

In this chapter, we present a compreensive description of the group structure: the tools used by its members to support their activities; how the newcomer training happens; and the general profile of FLUSP members. We paid special attention to the participants observed in this work, making comparisons between their profiles as contributors. We also extend the comparison to includes reflections of our profile as a FLUSP member and as FLOSS contributor. At last, an extensive study about their contributions will be performed, including an analysis of the patches sent and of the answers provided by the members selected to answer our questionnaire. We use those results to evidence that study groups can be used to achieve a greater member permanence in FLOSS communities and, thus, contribute to their subsistence.

3.1 FLUSP organization and principles

FLUSP organization follows a horizontal hierarchy, where all its members have the same voting powers, and there is no sense of seniority. Although each project developed under the group umbrella has at least one mentor, all decisions are taken democratically.

FLUSP is based on a set of four values[1]:

- Freedom of code and knowledge;
- Flatness;
- The belief sharing is paramount to the group and valuing the mutual exchange of learning among members ("passing the baton");
- Code reviewing as just as important as adding code.

The concept of member is still unclear in FLUSP. Despite the fact that the group held many meetings with the intention of creating a manifesto, no such thing has ever materialized up to this moment. Informally, though, anyone who has participated in a couple of meetings is considered a member and, consequently, has voting powers. Any member has the prerogative to propose the creation of a new project.

Although being a university extension group, it is not necessary to be affiliated to USP or any other institution of higher education to be a part of FLUSP. FLUSP is a group of students with no affiliation to any professor, although bureaucratically it has a professor responsible for the group activities, Alfredo Goldman.

[1] https://flusp.ime.usp.br/about

3.1.1 The tools that support group activities

Most information about the group is available in its official web page[2], such as tutorials, its meeting schedule and news about events. This website is developed using **Jekyll**, a static site generation tool written in Ruby. Jekyll provides an easier experience by allowing its users to write content in Markdown and then converting the raw text files into a static website. As FLUSP is comprised of volunteers, its resources are mostly limited to the knowledge its members possess. Since none of its members had much front-end web development when the site was created, Jekyll was chosen to facilitate its generation. In this way, the tool was able to decrease the learning curve associated with web page development and enabled the creation of the group website.

The files related to its web page are versioned as one of the group projects in **Gitlab**, a platform that provides a Git repository manager, along with a wiki page and a issue-tracking tool. Gitlab is one of the many pages on the web that offer control versioning through Git, and was chosen by the group for being FLOSS and for the cluster of features it provides in its free version. Besides its website, it is also in Gitlab that all the files under the FLUSP umbrella are stored, including the GCC mirror used by the participants who contribute to this project. Grounded on rules that value freedom of code and knowledge, all of the group projects on Gitlab are openly available to the public, with the exception of its website infrastructure project. The reason for that is because it contains critical data about the web page that the group still has not found out how to conceal.

Regarding writing permission to their projects, any member that manifests interest in getting involved can obtain it. One problem that troubles the group repositories, though, is the lack of organization pertaining on how contributions are performed. Most of the time, pull requests concerning the changed code are not created, and so the code is directly pushed into the master branch. Pull requests allow a contributor to suggest changes from one branch to be merged to another branch and are considered a good development practice, since they make it possible for users to discuss the proposed changes and potentially spot any existing issues on the code.

3.1.2 Newcomers Training Process

The group website provides a number of blog posts and tutorials with the aim of helping newcomers navigate more easily through the communities' cycle of contribution. Besides the production of materials and the group meetings, FLUSP also promotes events to attract contributors and provide FLOSS experiences for its members.

On March 26th, 2019, FLUSP promoted its first event, the GNU/Linux install fest. The FLOSS culture has a lot of force in IME, with Computer Science students being encouraged since their first year to use the aforementioned operating system. Many freshmen attended the install fest to have a GNU/Linux distribution installed in their notebooks. The event was considered a success, with about twenty installations being made that day.

In the following month FLUSP presented an introductory Git workshop. The workshop had the goal of teaching its participants the most basic Git commands, including how to use branches. It had around sixteen participants and was attended mostly by freshmen.

On May 18th, 2019, ocurred FLUSP major event to date, the KernelDevDay. KernelDevDay main goal was to help people who had never contributed to a FLOSS project make their first contribution. The chosen project was the Linux kernel, more specifically the Industrial Input/Output subsystem. The 21 participants were divided in nine teams and began work-

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

ing in the pre-selected issues. By the end of the day, fifteen patches were submitted to IIO mailing list, with one more having been sent afterwards. The feedback was overwhelmingly positive, with both participants and IIO maintainers praising the event outcome.

It was also possible to interact with many people involved in the Linux community during two events in which FLUSP members participated: DebConf 2019 and the linuxdev-br of the same year.

We can see the meaningful impact FLUSP was able to have in the Linux IIO subsystem. From the period of October 3rd, 2018 to November 17th, 2018, FLUSP contributions to this subsystem represented 20% of all the contributions sent to its mailing list. From January 4th, 2019 to April, 4th, 2019, this scenario happened again, since 19.4% of the patches sent to IIO were sent by FLUSP members.

3.2 FLUSP participants on FLOSS universe

Considering the FLUSP engagement on IIO, we observed the six participants of FLUSP with more contributions sent to this subsystem, in addition to contributions provided by the main author of this work. The seven participants observed in this study were named as follows:

- Participant 1 (P1): undergraduate student;
- Participant 2 (P2): master student;
- Participant 3 (P3): undergraduate;
- Participant 4 (P4): undergraduate student;
- Participant 5 (P5): master student;
- Participant 6 (P6): master stundent and co-advisor of this work;
- Participant 7 (P7): main author of this work.

P1, P3, P4 and P7 are undergraduate students in the final year of a Bachelor degree in Computer Science. P2, P5, and P6 are students getting their Master degree in the same course. Besides participating with patches sent through FLUSP, one member of the group - P1 - also contributed to the kernel project through GSoC.

To complement the analysis of each profile, we sent a questionnaire to the observed participants. P6 and P7 were the only participants that did not answer the questionnaire, since P6 and P7 were involved in the elaboration of this study. According to the answers given in the questionnaire, the mean age of the interviewees is of 24.2 years-old.

Through the questionnaire it was also possible to get acquainted with the development practices of each participant. P1, P2, and P5 showed to be people who prefer to develop contributions by programming individually. P3 and P4, on the other hand, stated a preference for programming in pairs. P7 is also a person that has an inclination to prefer to engage in pair programming.

3.2.1 Contributions to the Linux Kernel Project

There were sent to FLUSP mailing list a total of 277 patches. If we were to count only the final version of the patches, this number drops to 115 - meaning that 41.5% of the patches sent to this list were composed of final versions.

To analyze the contributions of FLUSP participants, we collected and classified the patches set to IIO in six different types: bug fix, code style, cover letter, documentation, enhancement and feature.

- *Bug fix* is a type of patch that intends to provide a solution for an existing bug in the code.
- *Code style* patch is one that aims to correct the writing guidelines used during the composition of a program source-code.
- *Cover letter* is created when a patch series is generated and usually has a description of the series' purpose and a short explanation of what each one of the patches does. Cover letters are not sent as a distinct e-mail when sent to the IIO mailing list. We decided not to take them into account when counting the percentages of each participants' patch types.
- *Documentation* patches are the ones pertaining the creation of material with information regarding the functioning of a driver or hardware. We also categorized as a documentation patch the ones with the objective of writing comments in the code to explain what it does.
- *Enhancement* patches offer minor improvements to a driver source code, but are not significant enough to be considered a new feature. At last, a feature patch is one that provides a meaningful modification to the code in a way that they can be considered a new attribute per se.

We were able to categorize the patches sent by people affiliated to FLUSP as following: 43 (37.4%) were enhancements, 37 (32.2%) were code style, 24 (20.9%) were documentation, 6 (5.2%) were features and 5 (4.3%) were bug fixes. The percentage of patches done together was of 32.2%, totalizing 37 patches.

All the seven people studied had a code style patch as their first one. This is according to the expected, because patches of this kind are considered the most simple ones and, as such, are a good way of familiarizing new contributors with the dynamics and contribution rules existing in the community. We also notice that the more experienced contributors possessed less patches made together. Creating patches along with other programmers can be seen as a way of transmiting part of one experience to the other. Therefore, it is to be expected that the less experienced contributors had more patches done in groups.

Additionally, it is expected that more experienced programmers have a larger amount of accepted patches among the ones sent to the IIO mailing list, and this scenario is exactly what happened within the collected data. Inexperience usually entails making more mistakes, so it is natural that the programmers that are in the community for a shorter amount of time have less patches sent to the IIO than their counterparties.

3.2.2 Becoming a Linux contributor

It is possible to notice that most of the final patches sent to FLUSP mailing list were also sent to IIO mailing list, since 96 (83.5%) of the final patches were sent there. Of all the final patches sent to the IIO, 71 (74%) were accepted by the list.

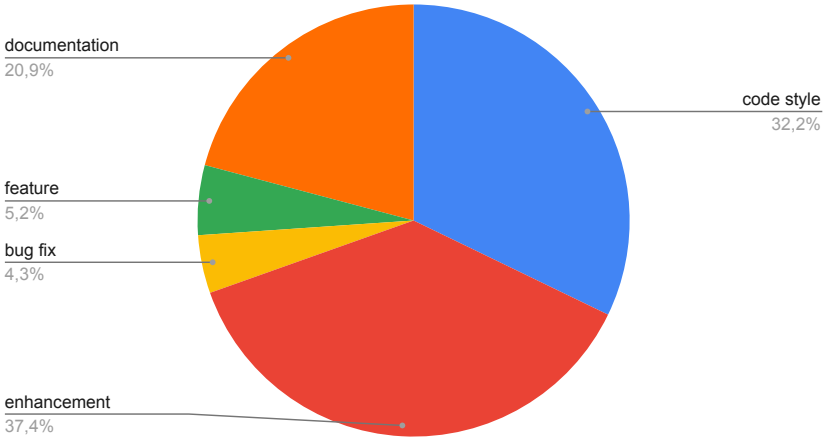
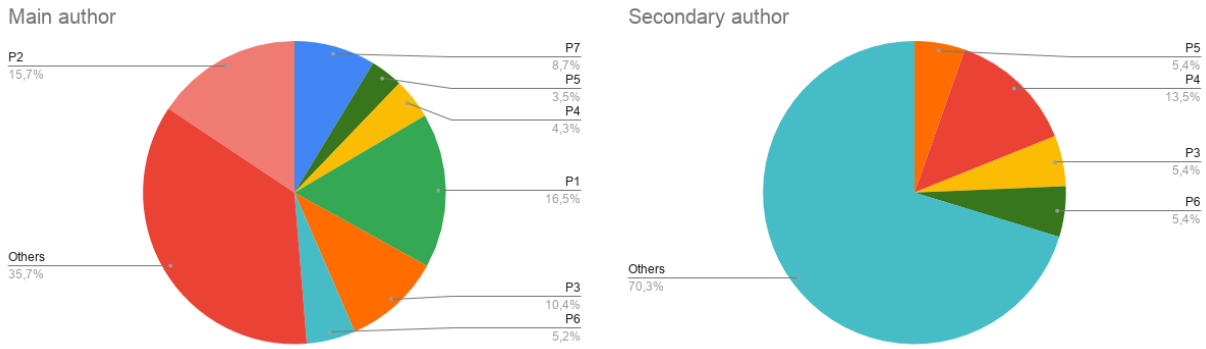


Figure 3.1: *Final patches by type*

There were 18 final patches sent the day KernelDevDay (KDD) took place, which represents 15.7% of all the final patches ever sent to FLUSP mailing list. Together these people contributed with 11 final patches, which amounts to 9.6% of all the final patches sent to the list. As those numbers can attest, KDD represented a significant success to the group mission of stimulating contributions to FLOSS projects. However, ten people that participated in this event have not made any other contribution neither before nor after through FLUSP mailing list. Therefore, KDD was not able to successfully insert those participants in the Free Software community.



(a) *Final patches by main author* **(b)** *Final patches by secondary author*

Figure 3.2: *Final patches by authorship*

Of the people studied, P1 was the one who sent most patches, with 19 patches as the main author and none as the secondary author, totalizing 16.5% of the patches sent as main author. The second person with most patches was P2, with 18 patches sent as the main author and none sent as the secondary author. This corresponds to 15.7% of all the patches sent as the main author. The third person with most patches is P3, with 12 patches sent as the main author and 2 sent as the secondary author, which is equal to 10.4% of the patches sent as the main author and 5.4% of the patches sent as the secondary author. The fourth and fifth people with most patches were P4 and us, the former with 5 patches sent as the main author and 5 patches sent as the secondary author (corresponding, respectively, to 4.3% of the patches sent as the main author and 13.5% of the patches sent as the secondary

author) and the latter with 10 patches sent as the main author and none as the secondary author (corresponding to 8.7% of all the patches sent as the main author). The sixth person with most patches was P6, who sent 6 patches as the main author and 2 as the secondary author, which corresponds, respectively, to 5.2% and 5.4% of the patches sent as main and as secondary author. The seventh person with most patches was P5, who sent 4 patches as the main author and 2 patches as the secondary author. This corresponds, respectively, to 3.5% and 5.4% of the patches sent as the main author and secondary author. The other participants sent in total 41 patches as the primary author (corresponding to 35.7% of the total) and 26 patches as the secondary author (corresponding to 70.3% of the total). Those numbers can be seen in Figure 3.2.

3.2.3 The diversity of contributions

Studying the final patches sent by P1, it is possible to categorize them as being 9 (47.4%) of enhancements, 6 (31.6%) of code style and 4 (21.1%) of documentation. In total, he sent 40 patches to FLUSP mailing list. Of all the final patches sent by P2, 5 (27.8%) were of documentation, 4 (22.2%) were of enhancements, 3 (16.7%) were of code style, 3 (16.7%) were bug fixes and 3 (16.7%) were features. In total, P2 sent 50 patches to FLUSP mailing list.

P3 sent, 8 (57.1%) were of enhancements, 3 (21.4%) were of documentation, 1 (7.1%) were of code style, 1 (7.1%) of feature and 1 (7.1%) were of bug fix. The member P4 developed 4 (40%) patches of code style, 3 (30%) of enhancements and 2 (20%) of documentation. He sent in a total of 22 patches to FLUSP mailing list.

Concerning P7's participation, it is possible to see that 6 (60%) of her final patches were of code style and 4 (40%) were of enhancements. As for P6, 4 (50%) of her final patches were of enhancements, 3 (37.5%) were of code style and 1 (12.5%) were of documentation. P5 sent 3 (50%) final patches of enhancements, 2 (33.3%) of features and 1 (16.7%) of bug fixes. In total, he sent 23 patches to FLUSP mailing list.

3.2.4 Different profiles of contributors

Despite being a development group, four of the seven participants observed seem to be a person that prefer to codify their contributions alone. Of the 19 patches sent by P1, 16 (84.2%) were developed without help from any other person and only 3 (15.8%) were developed with the help of another participant. In the questionnaire, P1 justified his preference for contributing individually by arguing that in this way he can do everything in his own time. He is one of the members with the largest proportion of patches sent to the IIO: all of his 19 patches were sent to that mailing list. He is a contributor with a high percentage of accepted patches, since of all the patches he sent, 15 (78.9%) received positive feedback and were included in the kernel code.

This characteristic also appears in the development of P2's patches, 16 (88.9%) of them were written without anyone else help and 2 (11.1%) were written with the help of one other person. In this case, with P5's. In the questionnaire, P2 justified his preference for individual contributions due to feeling he can be more productive this way. It is also possible to see that P2 has a large proportion of patches sent to the IIO. Of the 18 patches he sent, all of them (100%) were sent to this subsystem mailing list. This can be seen as a reflex of his experience in the group and of his abilities as a programmer. P2 also has the highest rate of accepted patches among the ones sent to the IIO. He has 17 patches accepted and 1 rejected, which represents an acceptance rate of 94.4%.

Two other participants who demonstrated to be more comfortable coding individually are P6 and P5. P6 developed 6 of her final 8 patches by herself, which represents a proportion of 75% of them being developed without any other person helps. Of the final patches sent by this participant, 6 (75%) were sent to the IIO mailing list and 2 (25%) were not. Of those 6, all of them were accepted by the IIO. She has the largest proportion of accepted patches among the studied participants. Few of P5 final patches - 2 (33.3%) - were developed together with another person. In this case, with P2. P5 expressed in the questionnaire a preference for individual contributions. He also possess the largest proportion of patches sent to the IIO mailing list, since all of his 6 patches were sent to it. His percentage of final patches accepted among the ones sent to the IIO mailing list showed to be similar to the other participants, being of 66.7%.

Another two observed participants and P7 seem to worry about pair-programming and co-development. P4 is an example of someone who prefers to develop patches with other people, since all his 10 final patches were developed with the help of another participant. He explained this preference in the questionnaire, saying he believed pair programming decreased the possibility of errors taking place. Their description of the group matches the definition given in this work, being of an extension group that aims to help students give their first contribution to a FLOSS project. He also has a high proportion of patches sent to the IIO. Of the 10 final patches he sent to FLUSP mailing list, 8 (80%) were sent to the former mailing list. The percentage of patches that were accepted among the ones that were sent is similar to the majority of the studied participants, being of 75%.

P3 is also more opened for co-development. Although he wrote 10 (71.4%) patches without co-authors, 4 (28.6%) of them were with someone else. All of his final patches were sent to the IIO and almost all the patches sent to this mailing list (13 of 14, or 92.9%) were accepted and had the code merged to the official repository.

P7 also demonstrates to be a person that prefers to develop patches with other people, since 6 of her 10 final patches (60%) were developed this way. Another thing that is possible to observe through the analysis of her patches is that most of them were different versions of the same patch. This reveals her inexperience in development, since if only the final patch versions were considered, P7's number of patches would drop from 40 to 10. It is also the reflex of her inexperience the proportion of her patches sent to the IIO, since only 11 (27.5%) of the total were sent to this list. Nevertheless, the acceptance rate of P7's patches did not show to be much different from the other studied participants. Of the 10 final patches sent to the IIO, 6 (75%) of them were accepted.

3.3 Breaking barriers with FLUSP

As for the reasons given for beginning participation in the group, the participants have mentioned: giving and receiving help to contribute to free software, developing research that benefits society and learning how to deal with a large codebase.

Three of the respondents stated they always put FLUSP mailing list in copy when sending contributions. They justified it by saying this strengthens FLUSP as a group and increases its visibility in the FLOSS community. The only person who said he did not put the list in copy is not actively contributing to the kernel and is currently the mentor in another project. As such, he is the one who reviews the patches related to this project.

The main difficulties pointed by the interviewees related to their engagement in a FLOSS project are: not knowing where to begin or what to do, difficulty finding documentation, and trouble understanding the source code and how it was organized. The way those problems were mitigated varied, with two members saying they first recurred to internet searches to

try to find the solution, one member saying his first option was the project IRC channel and another one saying he first recurred to FLUSP. Reading documentation and the community mailing list, and asking questions to the project maintainers were other ways those difficulties were solved.

All of the respondents agreed that FLUSP had helped them become FLOSS contributors. They credited the group as having rendered possible their insertion in those communities and for teaching them how to interact with its members.

Three of them assessed their level of experience in the main project they contribute to as "little experienced". This is a surprising result, as those three contributors have amassed a total of 110 patches sent to FLUSP mailing list. The respondent who assessed his level of experience in his main project as "very experienced" is not currently contributing to the kernel.

Lastly, there is a disagreement regarding the group structure. Although formally FLUSP has a horizontal structure, it has been pointed out by one participant that there is an implicit hierarchy with respect to the most experienced members of each project. There is, according to another participant, an absence of formalized responsibilities. This organization, or lack thereof, has been blamed for a certain lack of autonomy of its members.

Chapter 4

Final Remarks

By providing a support group to its members, FLUSP decreased the learning curve related to the ingress in a new project and the learning of the contribution process in the free software community. Since the group was already established, its participants can exercise a collaborative practice during its weekly activities and reunions. FLUSP was also able to provide its most experienced members with an engaging environment where knowledge sharing was possible. This factor can be associated with a lower dropout rate and higher permanence in the community.

Such student groups serve as an example to show how undergraduate initiatives can have a meaningful impact on a FLOSS project, therefore contributing to the maintenance and expansion of the FLOSS community. We also expect this work can motivate the creation of similar initiatives in other colleges.

By taking advantage of the lessons learned from the FLUSP experience, upcoming groups may improve their structure and efficacy even further. Pair programming can be seen as a tool to simplify the knowledge sharing among the group members. Internal reviewing decreases the number of faulty patches sent to a project's official repository, besides working to make the reviewer more acquainted with its source code. Blog posting is advantageous as a way for a participant to put into practice the skills they have acquired and is also useful for spreading knowledge beyond the group's scope. By exposing the writer's abilities, they are also helpful in case their author wants to apply for a job or a program, such as GSoC. Besides, the holding of events related to FLOSS projects can stimulate the interest of external people in the subject and potentially bring more members to the group.

Finally, this work can provide ground to the premise that FLOSS communities can use study groups as tools to promote the FLOSS initiative and improve the retention of its members. Further studies about this subject are encouraged as a way to better ground these findings. It would also be interesting to analyze why some of the group's initiatives - such as KDD - did not have the desired outcome of engaging new contributors in the long term in the FLOSS community, and what could be done instead to produce the desired outcome of sustained engagement in a FLOSS project.

Appendix A

Questionnaire on FLUSP's activities and contributions to FLOSS projects

1. Profile
    - What is your name?
    - What is your age?
    - What is your academic training?
2. About FLUSP
    - How would you define FLUSP?
    - In your opinion, what is the importance of FLUSP?
    - What motivated you to participate in FLUSP?
    - How would you describe FLUSP's organization/structure?
    - What activities does FLUSP perform?
    - What activities do you perform for FLUSP?
3. Your contribution profile
    - What FLOSS project(s) do you contribute in the context of FLUSP?
    - Do you contribute to any other FLOSS project in other contexts (without involving the group)? Which ones?
    - Did you contribute to any FLOSS project before participating in FLUSP? If so, which ones?
    - From your already made contributions, how do you rate your experience level in development in the main project you contribute to?
      ( ) Very experienced
      ( ) Moderately experienced
      ( ) Little experienced
      ( ) Inexperienced
      ( ) I don't know/I don't want to take a guess
    - What dificulties have you already experienced to develop to a FLOSS project?
    - How did you mitigate those difficulties?

- When you have any doubts, you usually first resort:
  ( ) To FLUSP
  ( ) To internet searches
  ( ) To tutorials/documentation
  ( ) To the project's community
  ( ) Other:
- How do you feel more at ease to develop a contribution?
  ( ) Individually
  ( ) In a pair
  ( ) In a group
  ( ) Other:
- Why?

4. Contributing through FLUSP

   - Did FLUSP help you to become a FLOSS contributor? If it did, how so?
   - Do you always put FLUSP's mailing list in copy in your contributions? Why?
   - Do you help your friends during the development of a contribution? If so, in what way?
   - What is the difference to you between contributing individually and contributing with FLUSP's support?
   - Do you usually share the FLOSS development knowledge acquired through the group? If so, in what way?

5. Comments

   - In case you want to make a comment about something that wasn't covered by the questions above, please share it here.

Bibliography

Corbet *et al.***(2005)** J. Corbet, A. Rubini and G. Kroah-Hartman. *Linux Device Drivers: Where the Kernel Meets the Hardware*. O'Reilly Media. ISBN 9780596555382. Cited on p. 6

Crowston *et al.***(2012)** Kevin Crowston, Kangning Wei, James Howison and Andrea Wiggins. Free/libre open-source software development. *ACM Computing Surveys*, 44(2):1–35. Cited on p. 1

DiBona and Ockman(1999) C. DiBona and S. Ockman. *Open Sources: Voices from the Open Source Revolution*. O'Reilly Media. ISBN 9780596553906. Cited on p. 5

Fitzgerald(2006) Brian Fitzgerald. The transformation of open source software. *MIS quarterly*, páginas 587–598. Cited on p. 6

Osterlie and Jaccheri(2007) Thomas Osterlie and Letizia Jaccheri. A critical review of software engineering research on open source software development. In *Proceeding of the 2nd AIS SIGSAND European Symposium on Systems Analysis and Design, Gdansk, Poland*. Cited on p. 1

Raymond(1999) Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49. Cited on p. 6

Scacchi(2010) Walt Scacchi. The future of research in free/open source software development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, páginas 315–320, New York, NY, USA. ACM. Cited on p. 1

Seaman(1999) Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572. Cited on p. 2

Silva(2019) Jefferson de Oliveira Silva. *Engagement in Open Source Software Projects via Summer of Code Programs*. PhD Thesis, University of São Paulo. Cited on p. 1

Tanenbaum and Woodhull(2011) A.S. Tanenbaum and A.S. Woodhull. *Operating Systems Design and Implementation*. Pearson Education. ISBN 9780133002058. Cited on p. 5