IT UNIVERSITY OF COPENHAGEN

Ph.D. Thesis

The Social, Organizational and Disciplinary Aspects of Quality in Free and Open Source Software Communities

Author: Adam Alami
Supervisors: Andrzej Wąsowski and Marisa Leavitt Cohn

A thesis submitted in fulfillment of the requirements for the degree of Ph.D. in Computer Science.

August 2020

Abstract (English)

Free and Open Source Software (FOSS) is an innovation model that does not rely on price or formal hierarchies nor alliance agreements. It may be described by the term "collective invention." A review of history reveals a number of examples of private-collective inventions, but most of these has not survive past the development of a dominant design, but FOSS has.

In FOSS, commitment to a community occurs because of the sense of autonomy contributors have, feelings of competence that grow as a result of successful contributions, and social relatedness. The FOSS production is a highly successful innovation model, and it survives the emergence of a dominant design, demonstrating it as a new innovation model. Collective invention in FOSS survives because of motivational aspects of contributors.

Although FOSS has the unique characteristics of being an information product, a user innovation, and the result of a highly modular design, these factors do not fully explain why FOSS produces high quality products. It is understood that quality assurance techniques, methods and tools are deployed in FOSS development process to control quality. However, these practices are not the only source of quality. For example, additional factors that may explain this phenomenon are conditions that foster pro-social intrinsic motivation. This dissertation asks how do social, organizational and disciplinary factors contribute to maintaining software quality in FOSS Communities?

I show that quality in FOSS communities is achieved when the environment facilitates some social, organizational and disciplinary *enablers* and *desired features*. I identified three enablers and two desired features. Enablers are qualities or capabilities that contribute to quality in FOSS. Desired features are intended capabilities, when achieved they created a desired effect which is maintaining quality. The enablers are (1) personal motivation for quality, (2)

governance for quality and (3) the ability to improve. The desired features are (1) active commercial participation and (2) retention of participants to sustain quality.

This is a mixed methods study. Mixed methods research is a methodology for conducting research that involves collecting, analysing and integrating quantitative (e.g., surveys) and qualitative (e.g., field observations, interviews) research. I conducted 82 interviews with FOSS contributors and maintainers. I carried out a Participatory Action Research project in the he Robot Operating System (ROS) community. I also conducted a survey with participants (N=387) from 15 FOSS communities.

Software quality is a difficult attribute to achieve. Software is produced with bugs for more than 60 years now. New solutions to prevent bugs continue to be developed in both research and commercial contexts. However, these solutions tend to focus on the technical aspects of software development, while, software development processes continue to produce bugs. This is, perhaps, caused by neglecting of other aspects of software development processes (i.e. social, organizational and disciplinary). It is time to broden the research attention to the non-technical aspects of quality. This dissertation demonstrates that quality has social, organizational and disciplinary dimensions that should be acknowledged, nurtured and studied further.

Software quality is achieved by technical and non-technical instruments. This implies that managing and implementing software quality necessitate also managing and nurturing non-technical mechanisms. For example, passion for developing software should be acknowledged, nurtured and rewarded. Another example, software development projects should embrace quality and pitch it as a fundamental believe rather than just merely a checklist.

Keywords: Software Quality, Free and Open Source Software, FOSS Communities, FOSS Quality.

Abstract (Danish)

Free and Open Source Software (FOSS) er en innovationsmodel, som er uafhængig af pris, formelle hierarkier og allianceaftaler. Softwaren kan beskrives som en "kollektiv opfindelse." Historien viser, at FOSS er en af de få kollektive opfindelser blandt privatpersoner, der er lykkedes med at fortsætte som innovationsmodel efter, at modellen har opnået status som dominerende design.

I FOSS opstår engagementet i et fællesskab, fordi bidragsyderne har en følelse af selvstændighed. De føler sig kompetente, og denne følelse vokser som følge af vellykkede bidrag og social samhørighed. FOSS er en yderst vellykket innovationsmodel. Det kommer til udtryk ved, at den har overlevet den dominerende designfase. Kollektive opfindelser i FOSS overlever på grund af bidragydernes motivation.

Selvom FOSS har et informationsprodukts og en brugeropfindelses unikke egenskaber, og er resultatet af et design opbygget af mange moduler, forklarer disse faktorer ikke helt, hvorfor FOSS fremstiller produkter af høj kvalitet.

Hvor FOSS' udviklingsprocesser bruger kvalitetssikringsteknikker, -metoder og -værktøjer, er disse er dog ikke de eneste procedurer, der anvendes til at sikre kvaliteten. Den høje kvalitet kan også forklares med andre faktorer som fx betingelser, der fremmer indre prosocial motivation. Denne afhandling undersøger, hvordan sociale, organisatoriske og faglige faktorer bidrager til at opretholde softwarekvaliteten i FOSS-fællesskaber.

Jeg viser, hvordan der opnås kvalitet i FOSS-fællesskaber, når miljøet fremmer nogle sociale, organisatoriske og faglige *enablers (katalysatorer)* og *desired features (ønskede egenskaber)*. Jeg har identificeret tre enablers og to desired features. Enablers er kvaliteter eller egenskaber, der bidrager til kvaliteten i FOSS. Desired features er tilsigtede egenskaber, der, når de opnås, skaber en ønsket effekt, som er at opretholde kvaliteten. Enablers

er (1) personlig motivation for kvalitet, (2) styring af kvalitet og (3) evnen til at forbedre. Desired features er (1) aktiv kommerciel deltagelse og (2) fastholdelse af deltagere for at opretholde kvaliteten.

Undersøgelsen er en mixed methods-undersøgelse. Mixed methods-forskning er en metode til at gennemføre forskning, der omfatter indsamling, analyse og integration af kvantitativ forskning (fx spørgeundersøgelser) og kvalitativ forskning (fx feltobservationer og interviews).

Jeg gennemførte 82 interviews med personer, der bidrager til og vedligeholder FOSS. Derudover gennemførte jeg et Participatory Action Research project (deltagende aktionsforskningsprojekt) i fællesskabet Robot Operating System (ROS). Jeg gennemførte også en spørgeundersøgelse med deltagere (N=387) fra 15 forskellige FOSS-fællesskaber.

Softwarekvalitet er vanskeligt at opnå. Der er blevet produceret software med fejl i mere en 60 år. Der udvikles løbende nye løsninger, der skal forhindre softwarefejl, både i forskningssammenhænge og i kommercielle sammenhænge. Disse løsninger har dog en tendens til at fokusere på de tekniske aspekter af softwareudviklingen, mens processen for udvikling af software fortsætter med at producere fejl. Det kan muligvis skyldes en negligering af andre aspekter i processen for udvikling af software (dvs. sociale, organisatoriske og faglige aspekter).

Det er nu på tide, at forskningens fokus udvides til også at omfatte de ikke-tekniske aspekter af kvalitet. Denne afhandling demonstrerer, at kvalitet har sociale, organisatoriske og faglige dimensioner, der bør anerkendes, plejes og undersøges nærmere.

Softwarekvalitet opnås ved hjælp af tekniske og ikke-tekniske midler. Det betyder, at når man arbejder med styring og implementering af softwarekvalitet, er det også nødvendigt at være opmærksom på og pleje ikke-tekniske mekanismer. Glæden ved at udvikle software skal fx anerkendes, plejes og belønnes. Softwareudviklingsprojekter bør desuden omfavne kvalitet ved at

fremhæve det som et helt grundlæggende princip, frem for at det bare er noget, der krydses af på en tjekliste.

Nøgleord: Software Quality, Free and Open Source Software, FOSS Communities, FOSS Quality.

Acknowledgments

A very special gratitude goes out to the ROSIN Project (grant No 732287) and the EU's Horizon 2020 for helping and providing the funding for this work.

I would like to thank my supervisors Andrzej Wąsowski and Marisa Leavitt Cohn for their consistent support and guidance during the project. Furthermore I would like to thank Peter Axel Nielsen from Aalborg University for his advise, during my stay abroad, and the collaboration on Paper D.

I wish to thank Raúl Pardo Jimenez for his help in conducting the quantitative data analysis for Chapter 10. I would like also to thank Yvonne Dittrich for starting this work and offering me the opportunity to do this Ph.D. at ITU.

I wish to acknowledge the support and great love of my family, my mother, Fatima; my father, Ahmed; and all my siblings. They kept me going on and this work would not have been possible without their emotional support.

Contents

Abstract (English) **ii**

Abstract (Danish) **v**

Acknowledgments **vi**

Contents **vii**

List of figures **xi**

List of tables **xii**

1 Introduction **1**

1.1 Context 1
1.2 Background 3
1.3 Terminology 5
1.4 Contributions 6
1.5 The Project Context 8
1.6 Outline 11

2 Problem Definition & Research Question **13**

2.1 Motivation 13
2.2 Defining Software Quality 14
2.3 Problems 19
2.4 Research Question 23
2.5 Theses 23

3 State of the Art **27**

3.1 Introduction 27

4 Quality in FOSS: The Case of the ROS Community (Paper A) **33**

4.1 Summary 33

4.2 Context and Motivation 33
4.3 Methods 34
4.4 Results 36
4.5 Contributions 39

5 Personal Motivation for Quality (Paper B) 40
5.1 Summary 40
5.2 Motivation 40
5.3 Methods 41
5.4 Results 42
5.5 Contributions 44

6 Affiliated Participation (Paper C) 46
6.1 Summary 46
6.2 Motivation 46
6.3 Methods 47
6.4 Results 48
6.5 Contributions 50

7 Continuous Improvement (Paper D) 52
7.1 Summary 52
7.2 Motivation 52
7.3 Methods 53
7.4 Results 54
7.5 Contributions 57

8 Governance for Quality (Paper E) 58
8.1 Summary 58
8.2 Motivation 59
8.3 Methods 59
8.4 Results 60
8.5 Contributions 62

9 Pull Requests Good Practices (Paper F) 64

9.1 Summary . . . 64
9.2 Motivation . . . 64
9.3 Methods . . . 66
9.4 Results . . . 67
9.5 Contributions . . . 69

10 Governing Pull Requests in FOSS (Quantitative Study) 70
10.1 Introduction . . . 70
10.2 Methods . . . 71
10.3 Subject Communities . . . 76
10.4 Findings . . . 81
10.5 Discussion . . . 88
10.6 Conclusion . . . 92

11 Discussion 93
11.1 Personal Motivation for Quality . . . 94
11.2 Active Commercial Participation . . . 97
11.3 Ability to Improve . . . 99
11.4 Governance for Quality . . . 101
11.5 Retention to Sustain Quality . . . 103
11.6 Chapter Summary . . . 109

12 Conclusion and Future Work 110
12.1 Future Work . . . 114

Bibliography 117

Appendices 130

A Appendix A: Paper A 131

B Appendix B: Paper B 140

C Appendix C: Paper C 152

D Appendix D: Paper D 164

E Appendix E: Paper E **187**

F Appendix F: Paper F **198**

G Appendix G: Pull Request Survey **209**

List of Figures

1.1 The Dissertation Outline 11

2.1 Extended Definition of Quality 18

4.1 Paper A Research Methods 36

5.1 Paper B Research Methods 41

6.1 Paper C Research Methods 48

7.1 Paper D research Methods 54
7.2 PAR4FOSS Framework 55
7.3 PAR4FOSS Instantiation for ROS 56

8.1 Paper E research Methods 60

9.1 Paper F research Methods 66

10.1 The Distribution of the Answers V27-V30 in Linux Kernel Community 81
10.2 The Distribution of the Answers V31 in FOSSASIA Community 82
10.3 The Distribution of the Answers V32 in FOSSASIA Community 83
10.4 The Distribution of the Answers V33 in the Coala Community 85
10.5 Bar chart which showing the probability of a community being protective, equitable and lenient. 86
10.6 Hypothesis 6 Test 87
10.7 FOSS Communities Pull Request Governance Model 91

11.1 Achieving Quality in FOSS Communities 94

List of Tables

1.1 List of my PhD Publications 9

2.1 Quality Definitions: Strengths & Weaknesses 16

2.2 Theses, Papers and Contributions 26

10.1 The survey questions relevant for this analysis 74

10.2 The PR Governance Styles as Defined in **Paper E** 75

11.1 PR evaluation practices and involved stakeholders 106

11.2 The problems and their corresponding solutions 108

1

Introduction

1.1 Context

Free and Open Source Software (FOSS) has attracted the interest of economic and social science fields as open source software is growing in importance in the software industry. Researchers note that open source is a paradox in economic theory applications. An examination of the history shows that FOSS production shares similarities to other examples in the past and present that can be summarized by the term "collective invention" [2]. In the nineteen century, the collective inventions were a phenomenon where rival firms shared pertinent information on non-trivial problems in development. The iron industry and Cornish pumping engine are examples [81]. Both formal and informal information was shared among firms. The cooperation was necessary because there was no prior knowledge on how to build a blast furnace, and the reputations of different engineers were viewed as useful to the effort. A disclosure of information did not harm the economy in the iron industry. The efforts were complementary, and added to the aggregate value of the product [81].

However, historically most of "collective invention" did not survive past the development of a dominant design, except in FOSS. Collective invention in FOSS survived because of motivation of contributors and the licenses that the communities follow. Commitment to a community occurs because of the sense of autonomy the FOSS contributors have, feelings of competence that

grow as a result of successful contributions, and social relatedness. The open source production is a highly successful innovation model, and it survives the emergence of a dominant design. It established itself as a new innovation model [85].

The occupational subculture of FOSS development is based on the shared beliefs of free software, free choice of work assignments, and cooperative work. Contributors are committed to contributing to the development and improvement of software. They uphold the principles of freedom of work assignments, freedom to use and modify software, and the goal of producing free quality software [36].

The success of any social movement depends on social acceptance, advantages for people, the creation of new social policies, implementation of new laws, and a shift in public policy. The open source software movement exemplifies these qualities. The principles of the free software movement are that anyone can run any software for any purpose, anyone is free to study and adapt the software for their own use, anyone is free to redistribute copies of the software, anyone can improve or alter existing software. The free software community and the open source communities differ in their views on licenses, but copyleft is an important element in both groups. Traditional copyrights do not fit with the computerization movement. The foundation for the FOSS movement is the belief that software should be free with all rights and developers should have free choice in assignments without a timeline or roadmap. The goals of the FOSS movement include the goal of building the FOSS community and immediate acceptance of new members to engage in cooperative work. The communities are self-managed informally self-management by the core developers and contributors using social control. Another important value of FOSS development communities is that of speaking the truth. Transparency and disclosure are important tenets of the community [35].

The motivation of members to join a FOSS community is the desire for control of their work. This "free choice" of assignments is important

to members of a FOSS community. The members identify with the work they produce, and it is a source of pride for them. Being able to pursue one's own passions and interests is a great motivator for members of the community [35, 87]. Members of the FOSS occupational subculture identify themselves as part of the community, as part of their self-image and their shared social identity that extends into their non-work identity [34]. The community members' self-image includes the traits of desiring "geek fame," a desire to build trust and reputation within the community, generosity of time, expertise, and source code, and the desire to create reliable and quality software [87, 97].

The "management" of a FOSS community is unique, but reflective of its principles, values, beliefs, and norms. The roles of members in a FOSS community have been described as an onion, with horizonal layers, rather than a vertical hierarchy. At the center are the core of developers, with the next layers consisting of the informal community managers, project managers, developers, and passive users making up the outer layer. Individuals generally move inward through merit and are nominated by another member of the community. Because the FOSS community members are often volunteers, the process of moving inward is through merit, such as facilitating others in their work, mediating conflicts, and solving problems in the community [54, 97].

FOSS is increasingly becoming a recognizable IT strategy [118]. Over half of the companies in Europe and the US are using FOSS for critical applications and 80% are using FOSS for application infrastructure [15]. Many FOSS projects have matured over the years to produce software of considerable size, complexity and some have seen generational changes.

1.2 Background

Free and Open source project refers to any software made public and open for others to modify. The idea of open source software began in the 1950s with

the release of SHARE, an IBM source code for IBM mainframes. In the 1970s, AT &T gave the Unix code to government and academic institutions, which basically made it free to interested parties. The modern open source software movement began with Richard Stallman in 1976, when he wrote EMACS and became a free software advocate. In 1983, Stallman developed the GNU operating system, a Unix-like system meant to use all free software. In 1985, Stallman began the Free Software Foundation, a nonprofit organization designed to develop, distribute, and modify free software. In 1998, the Open Source Initiative was founded, after the release of Netscape, to provide education, advocacy, and stewardship of free software. Open Source Initiative maintains a list of licenses for open source software [4, 87].

Originally software was perceived as a marketing incentive used to sell hardware. In 1964 with the IBM 360, the value of software was realized. Software was written for specific uses for specific industries, and sales of software grew. As software is capital intensive, a piece of software for a mainframe could cost as much as a million dollars. By the late 1970's as personal computer use grew, a new software industry emerged with prices being based on popularity rather than the power of the software. The code still had to be developed in a physical location as networks were too slow to share code development tasks. The personal computers were relatively inexpensive, but the software remained expensive because of the high capital required to develop it [10, 87].

Source code was no longer shared in the 1980's, but with the growth of the Internet in the 1990's, programmers in various locations could share the tasks of software development, and the FOSS community was born. Many developers shared code to create products. The cost of the code was only 10 to 15% of the total cost, and the rest of the cost of software was marketing, packaging, and supporting the product. Free and Open source software maintains nearly the same amount of expense on the code, but the support and packaging costs are much lower in the FOSS community [10, 87].

Understanding free and open source software (FOSS) includes knowing

what FOSS is not. It is not shareware, public domain software, freeware, or software viewers and readers without code. FOSS is associated with the hacker culture. Hackers are skilled professional programmers with norms for the hacker community. One of these norms is that an individual or small group controls the software, providing patches, fixes, and new releases. The original creators control the product or approve of another person taking on the role. They share the code in hosting platforms like Github and advertise it in social media. Another norm is that the developers of a software discuss the product on mailing lists and tracking issues systems. A final norm of hackers is that documentation is included with the FOSS products [7, 87].

1.3 Terminology

The term "open source" was proposed by Christine Peterson and voted on in a meeting of the Foresight Institute headed by Eric Raymond. The website opensource.org was instituted at the same time in 1998. The definition of open source is that it is software that has a non-restrictive license that does not allow it to be sold or given away and must include source code in the distribution. Open source software means software that allows modifications and derivatives of the work, and it must include the source code for modifications. Open source software cannot discriminate against any person, and it cannot restrict anyone from using the software. Rights apply to every user of the software, and area specific to the product [7, 61].

It is important to understand the terms "open source" and "free software". The legal implications of both terms are the same, but there are differences in the meaning of each term. Free software, a term initiated by Richard Stallman, is based on the personal ethic stance of an individual being free, as in free speech. On the other hand, Eric Raymond, initiated the use of the term Open Source, focuses on technical efficiency and neoliberalism, where the issue of freedom is less important than the Open Source movement. In his endorsement of the term Free Software, Stallman appeals to the pragmatist,

and the desires for growth of the Open Source Movement over the ethical concerns of the free speech.

While some individuals are proponents of open source and others of free software, only a few developers differentiate between these terms. I consider open source as a development method while free software is viewed as a political stance or social movement. However, open source project is a term that does not fit the traditional management view of projects because it is not a "temporary endeavor intended to create a unique service or product" [52], as project is defined. Instead, an open source project is an open ended endeavor and has a purpose of producing source code, especially for a software product. I propose the following pragmatic definition of an open source project: a group of people developing software collectively and making the final products available under an open source license. Perhaps the term should more appropriately be "open source collective" or "open source community" instead of the familiar "open source project." Throughout this dissertation, I use the acronym FOSS to include both philosophies, open source and free software. In addition, I tend to use the noun "community" instead of "project" to emphasise the collective aspect of the phenomena being studied.

1.4 Contributions

In addition to the traditional techniques, tools and methods (e.g. code review, testing) of achieving quality, quality in FOSS communities is achieved by additional social, organizational and disciplinary traits. I identified three quality enablers and two desired features that assist in achieving quality in FOSS software. Quality enablers are traits that makes quality possible. These are: (1) Personal Motivation for Quality, (2) Governance for Quality and (3) Ability to improve.

- ***Personal Motivation for Quality:*** I concluded that motivation allows FOSS contributors to excel in software development task (espe-

cially code review), which results in higher quality deliverables. High motivation for quality amongst FOSS contributors is visible to a greater extent. Achieving quality for FOSS contributor is internalized; it is an attitude part of contributors' behaviour by learning or assimilation.

- ***Governance for Quality:*** "without rules and controls, pull request process would be chaos. Quality is in our mind at every step of the process", this is how a participant asserted the significance of governance to achieve quality. I identified three styles of governance of the pull request process taking place in FOSS communities. These governance styles have in common is achieving quality and control.

- ***Ability to Improve:*** I have studied in depth the Robot Operating System (ROS) community. ROS is an exceptional FOSS case (which I explained in Chapter 4). Contributors in ROS have relatively little motivation for quality. To help the community align its quality practices with other FOSS communities, I implemented PAR4FOSS, a change implementation method derived from participatory action research. The endeavour showed that a FOSS community is able to improve its quality practices when the resources to assist in the implementation are available and motivated to do so.

The desired features are are intended capabilities, when achieved they created a desired effect which is maintaining quality.

- ***Active Commercial Participation:*** Commercial participation in FOSS is increasing. However, this participation sometimes is passive (i.e. inbound only). When it is the case, the sustainability of the community is impacted and consequently the quality deliverables. I advocate active participation of commercial entities to ensure an equitable use of the community resources and a sustainable growth.

- ***Retention of Participants to Sustain Quality:*** The pull request process is the entry point to the community and the environment where

the most interaction with the community occurs. Hence, I sought to understand what contributors perceive as fair treatment and judgement of their work. I identified seven good practices for the pull request process. When these practices are embraced in the process, they create good experience for the contributor.

Table 1.1 lists the publications, developed during my PhD project. Note that all these publications have been published, or in the process of being reviewed, in peer-review software engineering venues and journals.

1.5 The Project Context

This PhD project was carried out within the ROSIN project. The Robot Operating System (ROS) community, has attracted a worldwide community of users and contributors. One branch of ROS is ROS-Industrial, with a specific industrial application focus. Begun in 2012, ROS-Industrial Consortium has been able to collaborate with key players in the industry, such as ABB, Yaskawa, Siemens, John Deere, BMW, and Bosch. The goal of ROS-Industrial is to become the worldwide open-source standard for industrial robots. However, ROS-Industrial stakeholders have raised concerns regarding the quality assurance practice in ROS. They fear that the quality practices are not aligned with FOSS communities and software engineering best practices. The community sought assistance with their quality assurance, and the ROS-Industrial Consortium established the H2020 project ROSIN to enhance the ROS quality assurance practices (**Paper D**) and to promote ROS as a reliable robotic platform for industrial users. This work is one of the results of this community inspired project.

Paper A	Alami, Adam, Yvonne Dittrich, and Andrzej Wasowski. "Influencers of quality assurance in an open source community." In 2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 61-68. IEEE, 2018.
Paper B	Alami, Adam, Marisa Leavitt Cohn, and Andrzej Wasowski. "Why does code review work for open source software communities?" In Proceedings of the 41st International Conference on Software Engineering (ICSE), pp. 1073-1083. IEEE Press, 2019.
Paper C	Alami, Adam and Andrzej Wasowski. "Affiliated participation in open source communities." In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 1-11). IEEE. 2019
Paper D	"A Tailored Participatory Action Research for FOSS Communities" (under review)
Paper E	Alami, Adam, Marisa Leavitt Cohn, and Andrzej Wasowski. "How Do FOSS Communities Decide to Accept Pull Requests?" In Proceedings of the Evaluation and Assessment on Software Engineering (EASE), pp. 252-258. 2020.
Paper F	"The 7 Habits of Good Pull Request Evaluation" (under submission)

Table 1.1: List of my PhD Publications

1.5.1 The ROS Community

Although I collected data from several communities, the ROS community has the focus of this study. The ROS community is the result of a project begun in 2000 at Stanford University to integrate artificial intelligence (AI)

into software. Examples of this AI software are the Stanford AI Robot and the Personal Robots Programs. The ROS community created flexible, dynamic software systems for robotic use. In 2007, Willow Garage, a visionary robotics incubator started by Scott Hassan to accelerate the advancement of non-military robotics, gave significant resources to extend and create implementations of robotic software. Many researchers from various institutions and labs began to contribute time and resources to core Robot Operating System ideas and software, all under the open BSD license. Willow Garage became a for-profit company, and ROS is now maintained by the Open Source Robotics Foundation (OSRF). Over the years, the developed model has been named one of the strengths of ROS. This model allows any group to start their own ROS code repository that they maintain control and ownership of, and if they make their code public, they can receive recognition and credit. In this way, many can benefit from the open source software project.

ROS is an open-source meta-operating system that provides a flexible framework for writing robot software. It is a collection of tools and libraries that help to simplify creation of robotic platforms. The collaboration of researchers enables robotic products to be more robust. ROS provides a communications infrastructure as middleware. It offers asynchronous message passing, recording and playback of messages, request and response remote procedures, and a distributed parameter system. In addition to middleware communication, ROS provides robot specific features, such as standard message definitions, a robot geometry library, robot description language, diagnostics, pose estimations, localization, mapping, and navigation. The ROS toolset is one of its strongest features of the community, and this toolset enables debugging, plotting, and visualizing the state of the system with rviz and rqt tools.

There are over 3,000 ROS packages available for users and developers. These packages cover everything from proof of concept implementations to new algorithms for industrial drivers and capabilities. The ROS community has over 1,500 participants on the mailing list and 3,300 users on the Q&A

forum, with 22,000 Wiki pages with over 30 edits per day.

Recently the ROS community has embarked in an overhaul project called ROS 2 to reengineer and create a new architecture for the current software. ROS 2 is under heavy development and has attracted the participation of some major players such as Amazon and Intel. The ROS 2 project has currently become the focus of the community.

1.6 Outline

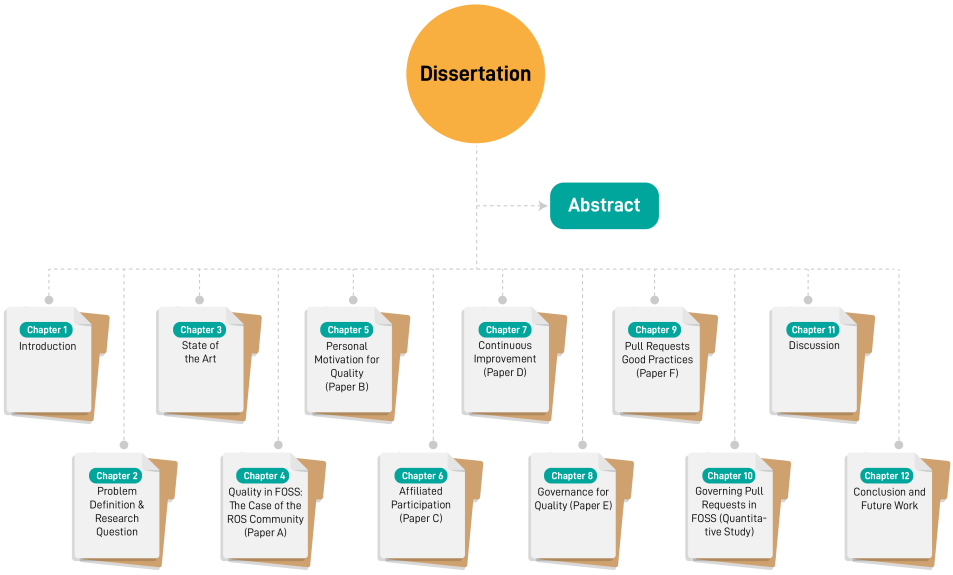

Figure 1.1: The Dissertation Outline

This dissertation is segmented into twelve chapters, as illustrated in figure 1.1. ***Chapter 1*** provides the context and the background necessary to introduce the reader to the dissertation topic. The next chapter, ***Chapter 2***, is an depth discussion of the problems and the research question underpinning

the work of this dissertation. ***Chapter 3*** summarises the current state of the art for the research question topic. ***Chapters 4, 5, 6, 7, 8 and 9*** are summaries of my publications. In ***Chapter 10***, I discuss the findings of the survey data analysis. The findings and the contributions of my dissertation are discussed in ***Chapter 11***. The conclusion and further work are highlighted in ***Chapter 12***.

2

Problem Definition & Research Question

2.1 Motivation

FOSS communities have demonstrated that they can deliver high quality and popular software. For example, Apache, runs 67% of the world's web sites [113]. Linux has become the most popular and versatile operating system kernel. It is used on super computers and web-servers, powering up cloud infrastructure, and controlling lots of mobile and embedded devices including all Android devices. The software were developed in a culturally and geographically diverse environment. Many developers volunteer because of need for pride, ambition, or sense of belonging to a community. In the FOSS world, a person is her/his reputation. All that matters is how well you do your job, how you write quality code. Once an individual earns her reputation, she works hard to keep it. FOSS communities are meritocracies, sometimes with a benevolent dictator at the top [113].

Typically, FOSS emerges and organizes organically [87, 110, 113]. Through the lens of transactional cost economics, FOSS communities are characterized as "bazaar" governance. They do not rely on employment contracts and no formal policy can enforce decisions. The difference between "bazaar"and market systems is clear [28]. Given this distinction, how these communities can sustain the quality of their software indefinitely is not yet understood.

2.2 Defining Software Quality

The quality of software has been referred to as desired excellence, value, conformance to specifications, conformance to requirements, fitness for use, loss avoidance, and meeting or exceeding expectations. To adequately define quality, it is important to trace the definition's roots, analyze the strengths and weaknesses of each definition, and describe the tradeoffs of accepting one definition over another.

Quality is a difficult concept to define in the context of software [74,99,103]. Software developers intend to create quality software, but the complexity of the effort makes it difficult to ascertain when the software is, in fact, a quality product. Inspection of software often consists of a systematic examination of the software program in detail, but "bugs" often still exist in software upon release [86]. The question that arises as the effort and inspections are made is exactly what quality software is, as well as how to achieve quality.

The five major views of quality are transcendent philosophical, product based, user based, manufacturing based, and value based [38]. Transcendent philosophical views of quality refer to innate excellence in the software. Product based views of quality address the quality of specific software products. The various customers have different perceptions of the software, but the aggregate perception is critical. A manufacturing view of quality is concerned with the supply side and conformance to specifications that fit the requirements of the software. Cost reduction is also important in a manufacturing view. Value based views of quality view the cost and price issues, and they are based on the belief that the higher the quality, the higher the price. Important issues in perceived software quality are features of the software, the bells and whistles; reliability; conformance, or how well the software matches the expectations; durability, how much use in the software during its life; serviceability, or repair issues; and aesthetics, how the software looks and sounds [38]. Table 2.1 presents the strengths and weaknesses of the definitions of quality. I conclude that the definition of quality is a continuum.

Software defects cost 40 to 1,000 times more to correct after release of the software [53]. Quality software is defined as without error, meets user needs, receives customer approval, ease of expandability, and reusability of code [20, 53, 56]. Quality software has a totality of features that bear on the ability to satisfy the stated and implied needs of customers. However, the metric that most defines software quality is defect control. There are two basic viewpoints about software quality. One of these is defines software quality as the process by which the software is developed, and the other is the evaluation of the software by assessing the end product, focusing more on usability [58].

FOSS appears to operate in unconventional fashion that quality software would appear to be difficult to create, and the lack of mechanisms to ensure quality, along with little architecture planning and no management teams contribute to this assumption. The quality of FOSS is attributed to its peer review process, fewer constraints on developers, developers motivated to create quality software, and collaborative communities [74]. Another reason for the high quality of FOSS is the openness of the entire project, including code and documentation, which allows for more feedback that can be used for improvements of these artifacts [74].

Paper A revealed that software quality in FOSS is not a single, easily definable entity, but a complex, interconnected concept fluctuating between the stakeholders. The study concludes that quality cannot be defined in isolation of its social and organizational environment. Quality is a difficult concept to define and examine in software development [74]. Hence, I decide to work with the assumption that the definition is subjective and an ever evolving concept. The aim of my dissertation is not to revive the debate of the definition of software quality, but to steer attention to the environment where quality is created. Quality is more than correctness and meeting specification; it is also achieved socially and organizationally, as this dissertation will argue. My studies participants tend to agree. For example:

Quality Definitions	**Strengths**	**Weaknesses**
Value	The concept of value incorporates multiple attributes, it focuses attention on a firm's internal efficiency and external effectiveness, and it allows for comparisons.	Weaknesses include difficulty extracting individual components of value, and it is questionable what is included in value.
Conformance	Strengths are the ability to conduct precise measurement, its relationship with increased efficiency, and its ease in use in global strategy.	Weaknesses are that consumers do not know or care about internal specifications, it is an inappropriate definition for services, it potentially reduces organizational adaptability, and specifications may quickly become obsolete in rapidly changing markets.
Meeting expectations	It has the strengths of evaluating from the customer's perspective, it is applicable across industries, it is responsive to market changes, and it is an all-encompassing definition.	Weaknesses are that it is the most complex definition, it is difficult to measure, customers may not know expectations, short-term and long-term evaluations may differ, and there is confusion between customer service and customer satisfaction.

Table 2.1: Quality Definitions: Strengths & Weaknesses

"I'm not sure there is a specific way to assess quality. We can read through the code and we know good code from bad code. It is quite subjective. However, in our community, there is a requirement for a minimum 3 reviewers to approve code. That makes it objective."

(Participant 27, Coala Community, Paper E)

2.2.1 An Extended Definition of Quality

The strongest quotes from my data that resonated with me are these statements from a project manager with over 30 years experience in FOSS software development and adoption and a senior Linux engineer. They stated:

"You cannot talk about quality in open source in isolation of the environment we work in. Quality in open source has a lot to do with the people, their motivation and morals. It is also contributed to the nature of open source, how it operates and controls the software development processes."

(Participant 25, Linux Community, Paper E)

"We obviously have our tools and processes to assure quality. But quality is more than that! Your code quality is your reputation and pride. We all strive to deliver the best code possible and we learn from the best."

(Participant 15, Linux Community, Paper B)

Based on this and other empirical evidence I encountered during my studies, I argue that quality definition can be extended beyond the current debate. Quality has other dimensions. I identified four non-technical dimensions of quality, (1) a social contract, (2) disciplinary customs, (3) organizational practices and (4) ethical obligations. This is illustrated by figure 2.1.

- **A social contract** is the implicit agreement in the community amongst its members. Quality becomes a social obligation toward the community. Members of the community collaborate to produce quality for the benefit of the community. For example a participant stated, "... In

Figure 2.1: Extended Definition of Quality

open source projects we like to achieve higher code quality. Because its open source and we will need to get good quality code..." (Participant 15, DuckDuckGo Community, Paper E). This believe was echoed by many of my participants. They talk about quality as if it was a duty toward the community.

- **Disciplinary customs:** This is the set of measures taken by the community to regulate and control every software development activity to create order and direction. For example, the Coala community has a rule that every pull request (PR) must obtain the consensus of three reviewers prior to the PR being considered for merger.

- **Organizational practices:** Practices related to organizing the community and the activities related to software development. For example, Paper E shows that the studied FOSS communities govern their PR processes using a combination of social norms and software engineering principles.

- **Ethical obligation:** In FOSS, quality becomes an ethical category,

the "right" and "wrong". Quality is "right" and mediocrity is "wrong" in FOSS behavioral system. In some communities this is taught through mentoring. Other communities (e.g. Linux) ingrain quality as a ethical obligation through harsh feedback and rejections.

2.3 Problems

The software quality problem has been a leading issue for the software industry for 40 or 60 years now. The problem persists not due to lack of methods, tools and processes for quality. There is an exceptional amount of knowledge on how to produce software of high quality. But this knowledge has focused on techniques, tools and methods to assure the internal quality of the software (i.e. the code) and the output of the software (i.e. fit for purpose and meeting the business requirements). Testing merely verifies that the software performs correctly under a wide range of operational conditions. Focusing solely on testing ignores the major recognized source of errors: the human in the process. With this dissertation, I want to attract attention to the social and organizational aspects of quality. I claim that the emphasis solely on the technical and control aspects of managing quality ignores the social, and organizational aspects of quality. The human element is critical to quality [91, 112]. Most quality concepts and standards come from the manufacturing field, but software quality requires a multiperspective [24]. Quality includes the human aspect, as well as technical aspects, and the culture must be appropriate to creating quality [84, 91, 111, 112].

> ***Problem I.*** *There is a need to understand the social, organizational and disciplinary aspects of quality in software engineering. However, this direction received little attention in software engineering literature.*

Although empirically it has not been proven yet that open source tends to produce better quality software than its proprietary or alternative counterparts, the success of FOSS product is a testimony to its quality. The

"Coverity Scan Open Source" report, which measures the quality of FOSS code, finds that the density of code defects (the number of bugs per 1,000 lines of code) is smaller for FOSS than for proprietary software [66]. Quality in FOSS communities is definitely achieved by the combination of good software engineering practices and non-technical aspects (i.e. social and organizational). For example, in FOSS communities, a developer's reputation is important. Contributors who produce higher quality work are more likely to be offered to work at higher positions in the community [33]. Cai and Zhu show that a developer's reputation is determined by the individual's coding quality, the commitment behavior, community experience, and collaboration experience [8]. When a developer is known, he or she will work to make high quality software so that he or she can build a reputation for high quality software development. Many developers describe the job of programming as "joy". Part of the joy is that developers can pick the projects that they wish to work on. In addition, the developer is not put on a tight schedule of development, but he or she can work at his or her own pace. The developer in open source software communities can decide what he wants to work on and how long he will work on it. Another joy of programming is that the typical programmer loves to learn, and open source development projects can be an opportunity to learn from other developers. Linus Torvalds decided that the best way to solve complex coding problems was to open them up to many developers and let the best code win. Developers can enhance their own skills by examining the code of the best developers. [87]

Open source motivations have a strong ethical component and allows for greater transparency of process. Open source proponents claim that passion about software development and producing software for personal satisfaction result in highly productive environment and high-quality software [106, 114]. Transparency of a development environment in an open source projects provides visibility of popularity and proves liveness of the project. Such transparency also enables evaluation of contributions and consequently merits are attributed to those who deliver quality code. How these social traits influence achieving and maintaining quality in FOSS is inadequately

understood.

> ***Problem II.*** *There are anecdotal evidence that quality in FOSS communities is achieved by more than techniques, tools and processes. However, how these social aspects interact with achieving quality is not demonstrated empirically.*

FOSS communities are informal virtual organizations, they do not rely on explicit hierarchical organization and structural authority. Control is exercised via a self-organized meritocracy that governs the community [35,36]. They challenge the established norms of the technical world. Part of the challenge is economic, and part is methodology. With the code open to all, revenue cannot be made from the protected code. A FOSS community is usually created by a team outside of a corporation that was distributed all over the world. The goal of the software is to provide a solution to a need that has no mandated schedules, no predefined work plans, and no deadlines. There is no bureaucratic governance in place that can veto new ideas and projects. Many open source teams have a small core of developers. Another distinction is the chain of command. The team leader is usually the one who had the idea for the project, and the qualification for most team leaders is a dedication to seeing the job completed. The final distinction is the review process. Most work is done in the open, and test versions are made public, where "many eyes" can provide feedback during the process [87].

"Various definitions and statements on quality lead to the fact that quality depends essentially on people." [84] Current challenges facing software development performance improvement are largely organizational and not technical in nature [84, 91, 111, 112]. Identifying defects is one way to improve quality, but improving the culture that exists is important [112]. How these unique organizational traits influence maintaining quality in FOSS community is not explored yet.

> ***Problem III.*** *FOSS communities are recognizably different socially and organizationally. However, how these social, organizational and disciplinary aspects interact with achieving quality is not understood.*

Not all FOSS communities have reached the same level of adopting quality practices. Some luck enthusiasm and motivation for quality. These communities need to align their quality practices with other FOSS communities. However, how to change a community from an "as is" state to a "to be" state is not well understood. FOSS communities are culturally, socially and organizationally distinctive organizations. Not all FOSS communities have a governance mechanism process in place; and if they do, it is not understood how these governance structures play a role in introducing change to the community.

> ***Problem IV.*** *How can a community manager or leader steer the quality of the community to a positive direction? Communities with immature quality practices need to go through a change process to enhance their conditions. However, the literature does not propose a change method tailored to FOSS communities distinctive traits.*

There appears to be a trend for more FOSS adoption in corporate settings [109]. Reduced software acquisition costs and the ability to innovate and resources are benefits that attract commercial organization to FOSS [47, 105]. Assessing the quality of the software is a fundamental part of the software acquisition process. However, the value system of FOSS development often emphasizes different aspects of quality due to different underlying assumptions and a different working method. The differences may be subtle in some cases and more explicit in others. FOSS software quality cannot be assessed by a checklist. It has a human, organizational and disciplinary dimensions that should be examined as part of assessing FOSS quality.

Problem V. *What are the positive signs of quality in FOSS communities? This is one of the questions that commercial adopters should ask when selecting FOSS software. However, these positive signs are not known.*

The adoption of FOSS software is not a simple transactional operation. It binds the company to the FOSS community as the company becomes dependent on software updates and future releases. Some companies choose to participate passively (i.e. inbound only)

2.4 Research Question

This dissertation addresses the following question:

How do social, organizational and disciplinary factors contribute to maintaining software quality in FOSS Communities?

"Social" in this context means the qualities and behavior of people. "organizational" is the act of organizing a FOSS community and the activities relating to it. "Disciplinary" implies enforcing a particular discipline. In this context, it's the set of measures taken by the community to regulate and control every software development activity to create order and direction.

2.5 Theses

Based on the above research question, I propose the following theses:

- **T1** A number of human and social aspects create a psychological and social environment that drives the contributors to excel and collaboratively produce high quality code. This drive to excel and investing care during the review process contribute positively to the software quality in FOSS communities.

- **T2** Active commercial participation in FOSS enhances the sustainability of quality. The "free-riding" phenomenon is prevalent in some FOSS communities. I argued that passive participation strains the community's sustainability, it leads the community into regression which hinders growth and ability to innovate. This consequently has impact on the maintenance of software quality. Hence, I advocate for active participation. Active participation strategy combines pecuniary (related to competitive assets and producing rewards) and non-pecuniary (related to non-competitive assets without immediate rewards) contributions.

- **T3** Obviously, not all communities are equal. Some communities are struggling to align their QA practices with similar FOSS communities. In this instance, the community should implement a continuous improvement endeavor to develop its quality practices. I argue that FOSS communities have the ability collaborate and invest effort in implementing change. To be able to evolve and continuously improve quality practices is a demonstration that FOSS communities can pursue achieving software quality. I designed and executed a participatory action research method (i.e. PAR4FOSS) tailored for FOSS tailored for FOSS cultural, social and organizational distinctiveness.

- **T4** Software engineering principles are not the only criteria applied in pull requests (PR) evaluation; social and strategic criteria are also of high importance. Software changes are not taken lightly in FOSS communities. Each community adopts a governance style to oversee the quality of the suggested changes to the code base. Having in place a controlled form of behaviour or way of working contributes to maintaining software quality.

- **T5** FOSS software development is a highly social activity. The literature suggests that the PR process has an unpredictable outcome, which deters contributors from further participation. I argued that the contributor's journey should be enhanced by good PR practices to improve contributors retention. A critical task of sustaining commons

is to ensure an adequate number of participants, which are the most valuable resource. Hence, I suggest a set of good PR evaluation practices (e.g. engagement with the PR and communication) to encourage participation.

In summary my Ph.D. demonstrates that software quality in FOSS communities is maintained by additional social, organizational and disciplinary values and qualities. I showed evidence of non-technical quality competency in FOSS communities. I identified three enablers (i.e. motivation for quality, PR governance and continuous improvement) and two desired attributes (i.e. active participation and retention) to maintain software quality. They act as catalysts for quality. Table 2.2 shows the argument how my publications address the theses listed above.

Theses	Type	Publication	Contribution
T1	Social	*Paper B*	Quality is facilitated by motivation. Managers should create a climate of enthusiasm and motivation. Management should recognize merits of the individual and the group.
T2	Organizational	*Paper C*	Paper C highlighted the need for constructive participation in FOSS by commercial organization. Active participation sustains quality in FOSS. When passive commercial participation is a dominant behavior in a community, quality suffers. The balance of the FOSS ecosystem is compromised as the exploitative practices regress the community.
T3	Organizational	*Paper D*	FOSS communities should demonstrate their ability to improve when their quality practices are not aligned with other FOSS communities. This ability to change is a desired feature for quality. It is a testimony that FOSS culture and work habits can be influence to a positive direction.
T4	Disciplinary	*Paper E*	Paper E shows that process governance enables quality. Although communities choose different ways to govern their pull request process, they have the same aim, which is assuring quality.
T5	Social	*Paper F*	Paper F shows that quality needs highly motivated resources. The pull request process is a critical venue for contributors' experience. It exposes them to the community norms and rituals. Ensuring a good contributor' experience is an investment in retaining contributors.

Table 2.2: Theses, Papers and Contributions

3

State of the Art

3.1 Introduction

Software development can be considered as a fundamental social process embedded within organizational and cultural structures [1,25,44]. These social structures enable, constrain, and shape the behavior, knowledge, programming techniques, and styles of software developers. Understanding how people work together to build software is critical since software's importance in our society is matched by the difficulty encountered in its development.

Software engineering is concerned with developing software that has the quality of satisfying both functional and non-functional needs, overcoming internal and external constraints, while maintaining usability, compatibility, portability, reusability, and adequate documentation. Doherty claims that 90% of software failures are non-technical problems, but they are related to social and organizational features [32]. Therefore social, personal, and group factors are as important to understand as are methodologies and automation. Social dependencies and networks among developers allow them to interact and coordinate with one another to create software.

The scope of this chapter is the review of the state of the art of the social, organizational and disciplinary aspects of quality in FOSS communities. However, my search shows the related work is almost nonexistent. Therefore, I elevated the scope to cover software development instead of being restricted

to the quality part of the whole process. The literature of this topic is separated into two streams of work: (1) social aspects, and (2) organizational aspects.

3.1.1 Social Aspects

Social aspects are the commonalities among people within a specific culture. Social aspects may include the following: language, norms, rules, team members interactions, and group behaviors. It also includes how developers work together to produce software. Cain and colleagues explain that software engineering too often pushes social concerns aside, perhaps dismissing them as "unscientific" and therefore as being ill-suited to a so-called engineering discipline [9].

Software development is a socio-technical process [9, 67, 94, 96]. There is considerable evidence that software development processes are influenced by social and psychological factors [25, 44, 70, 94]. Programmers do not exist in isolation. They are usually part of an organizational structure that embraces believes, norms and values. Software development activities require coordination. Inevitably, during coordination and communication, the developers are influenced by each others' domain knowledge, programming techniques, and styles. Such influence can be uncovered in software repositories and found in the structure of the software artifact itself [27]. Therefore, software development can be considered as a fundamental social process embedded within organizational and cultural structures [25]. These social structures enable, constrain, and shape the behavior, knowledge, programming techniques, and styles of software developers [44].

Software development is a predominantly social activity. It is important to view software development groups, departments, and corporations as social bodies. The essentially human nature of customer interactions, programmer creativity, and programming team dynamics demand that we deal with the social side of software production enterprises [25, 44].

Software development is a labor intensive project, and the majority of software is developed by teams. Therefore, the dynamics of their interactions play a major role in the success or failure of a software project [94]. Rosen concludes that individual behaviors affect the environment, and the environment influences the behavior of individuals. This is expressed as pride in the product, a desire to produce the best product possible, and an anxiety about defects in the product [94].

Chong, et al. examine the socio-cognitive factors of pair programming [16]. Other studies have shown that pair programming gives better designs, more compact code, and fewer defects. In addition, programmers exhibit greater confidence and enjoyment when the product was the result of pair programming [45, 72]. In examining the pair programming technique, Chong, et al. observed that work is split into two roles, one of the driver, the person at the keyboard, and the navigator, the active observer and monitor of code. They collaborate on all aspects of the development. They are in constant communication, asking and answering questions of each other. They may switch roles frequently. The simplest reason that this technique works is that two people make better design decisions than one person can. Two individuals will have overlapping, but not identical, sets of information; the collaboration is a mutual apprenticeship, where each learns from the other; the collaborative design requires the negotiation of a shared understanding and mutual orientation; the negotiation process requires that programmers share goals, plans, decisions, and actions, which leads to a more through exploration of design options. This production, verification, and affirmation leads to increased confidence in the programmers and vet flawed design ideas by the pair. The navigator can look for missed cases and typographical errors, as well as think ahead of the code being typed at that moment [16].

Marshall and Webber examines taboos that unconsciously influence software developers. These taboos operate on the work of programmers in the form of lore, which is the way programmers learn and communicate, and magic, which is the process of implementation. Lore came about with the

design pattern community, who by nature are classifiers and cataloguers. Design patterns provide little templates for solutions that are common in creating software. Many developers learn by watching and doing, and the absorption of taboos occurs this way as well. Some bits of code get reused, so these bits become "folk tales" [70].

Besides lore, magic is another taboo found in software development. Sympathetic magic is the solving of problems in programming by simulations, by constructing models. Programmers have their own styles, yet if they find a way of doing something that works well, they stick with it, even if it seems irrational. Sympathetic magic also takes the form of naming of functions, that is calling them after what they want them to do rather than what they actually do. They also see certain actions as a talisman that protects them from errors, and this talisman becomes a superstition, or a charm. The charm can become a reflex to developers [70]. From this analysis of lore and magic in software development, the researchers state that encouraging the spread of programmers' lore could be a good thing. Secondly, there is more to understanding programmers and programming than looking at processes and outputs. Social interaction, which is where these taboos are acquired, can play an important role in how programmers' knowledge develops [70].

This literature focuses on the aspect of collaboration amongst software developers. It is an interesting social aspect, but there is a need to expend our research to cover other aspects and with paying more particular attention to others facets of software engineering, e.g. achieving quality, requirements engineering, etc. I asked, how do social, organizational and disciplinary factors contribute to maintaining software quality in FOSS Communities? I concluded that personal motivation play a important role in achieving quality in FOSS.

3.1.2 Organizational Aspects

Organizational (e.g., structure of organization, management strategy, business model) aspects of a software development team follow Conway's law [18], which states that software reflects the organizational structure that produced it. The division of labor and software architecture reflects the structure and social climate of the software team that created it [18].

Research has been preoccupied with technical issues at the expense of people and organizational issues. Organizational issues are any distinct area of the interface between a technical system and either the characteristics and requirements of the host organization or its employees, and organizational issues are more important than ever in the successful development of information systems [32].

Doherty and King conducted a survey on the organizational issues affecting software development. The majority of respondents perceived organizational issues as being the most important issue in software development, but these issues are rarely addressed in an organization. Senior IT managers coming from larger and more sophisticated organizations are more likely to address organizational issues. In addition, senior IT managers overseeing the implementation of packages might be more likely to address the issues than others [32].

Organizational issues are given low priority because respondents feel that the solutions are intangible, ambiguous, and politically sensitive. Tight time and cost constraints also come into effect. Solutions that have been suggested are the need for integrated IT and user development teams, organizational ownership of systems throughout the development process, formal change management programs, and the need for both user and IT communities to work more closely together [32].

There is limited attention to the organizational aspects. My dissertation shows strong influence of these factors on the overall growth and sustainability

of FOSS communities. I concluded that commercial participation influences the sustainability of the ROS community which cascaded to affect quality related tasks. I also showed that a FOSS community ability to implement continuous improvements program are fundamental to sustaining quality. We also need to distinguish between FOSS development and closed-source development when studying organizational aspects. They both have different organizational aspects. Their organizational structures and cultures are fundamentally different.

This literature review shows limited work in the area of social, organisational and disciplinary aspects of software development. This has been explained by few authors. It's either because the research community feels that the subject is "unscientific" [9] or simply because there is an over emphasise on the engineering aspects [32]. This dissertation is a testimony in favor of this topic. It shows that there is a profound correlation between achieving quality in FOSS and social, organizational and disciplinary aspects.

4

Quality in FOSS: The Case of the ROS Community (Paper A)

4.1 Summary

This research study was designed to investigate quality assurance practices in the ROS community to identify factors affecting those practices and recommend ways to overcome challenges and intensify positive practices in the community. Six factors were found through mixed methods research that affect quality practices. These are participation motives, priorities of the community, the meritocratic culture, sustainability, complexity, and adaptability.

4.2 Context and Motivation

The motivation behind this study came about when I joined the ROSIN project. The project planned interventions activities to enhance the ROS community quality practices. The ROS environment (e.g. culture, attitude toward quality, social norms, etc.) was foreign to the project. To ensure successful interventions design and execution, I suggested to expand a "preliminary study" done by the project [31]. Although the study disclosed extensive technical knowledge about achieving quality in an open robotic community,

it lacked an understanding of the social and organizational aspects to ensure successful execution of the envisaged interventions.

FOSS communities differ from traditional closed source teams. There exist a vast amount of different norms and believes that facilitate the development of open source software. These norms and believes impact the software quality. This effect is not well understood yet. To meet this objective, I opted for semi-structured interviews with ROS developers. Semi-structure interviews are useful to obtain detailed information about personal feelings, perceptions and opinions which was not sufficient by examining online resources [46, 78].

The Robotic Operating System or ROS Community is a large community that develops complex software for robotics. Many off the developers are not software engineers. They are largely engineers from other disciplines (e.g. electrical engineering, mechanical engineering). Because of the needs of the collaboration of different technological capabilities to develop the software, ROS attracted contributors from various disciplines. Its Wiki platform is a busy site with 1.4 million visitors a year with nearly 7,000 registered users. There have been over 13.4 million downloads of the software. Over ten years, ROS has become one of robotics' *de facto* standard operating systems. The ROS community has different attributes than more commonly studied communities, such as Linux and Mozilla. The ROS community is unique in that it produces software components for robotic systems. It is a diverse multidisciplinary community with developers from many fields who have experience as package developers, students, chief technical officers, and other fields.

4.3 Methods

This dissertation is a mixed methods study. As shown by Figure 4.1, Constructivism is the belief that we construct our view of the world based on our perceptions of it. Positivism is the belief that science is seen as the way

to get at truth, to understand the world well enough to predict and control it. Positivism is the idea that observation and measurement is the core of scientific endeavor and the scientific method is the way to learn about the world [115]. A positivist approach to a research topic is usually related to quantitative methodology, while a constructivist approach is related to a qualitative methodology.

Positivism is often referred to as objectivism, and constructionism is referred to as subjectivism. Positivism is based on the belief that the world can be measured based on facts, while constructivism is based on the mind's interpretation of what is observed. The frame of reference is important to a constructivist approach to research, while it is not important to a positivist approach in research [115].

The research strategies of this dissertation are stemmed from these two distinct philosophical stands, constructivism and positivism. This choice is rooted in my believe that every research question has the correct corresponding research method. One research strategy does not fit for every research question. I selected my research strategies (i.e. Thematic analysis, participatory action research, case study and survey) based on a particular need to address the research questions, I asked.

This paper (**Paper A**) is a case study of the ROS community using semi-structured interview for data collection and grounded theory (GT) for data analysis [12, 13]. Ideally, grounded theory is applied throughout the research process, that is, from conception of research questions to concurrent sampling and data analysis. However, in this case, I used GT for coding procedures after all of the data have been collected.

This research study implements a mixed methods research method consisting of interviews, a virtual ethnography, and community participatory observations. The interviews were semi-structured interviews with ten participants, which, along with the other tools, were used to collect data that

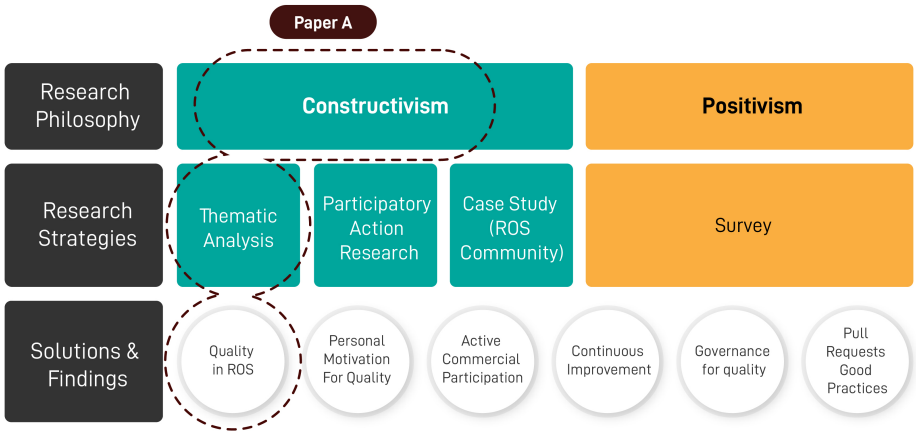

Figure 4.1: Paper A Research Methods

was organized and interpreted. The data was analyzed through open coding, focus coding, and theoretical coding [12, 13].

4.4 Results

During the data analysis, I identified six forces affecting the implementation of quality practices in the ROS community, namely participation motives, community priorities, meritocratic culture, sustainability, complexity, and adaptability.

One of the forces affecting implementing quality measures in the ROS community is FOSS ideology [108], or a mental model framework that is used to interpret the environment and steer decisions. One aspect of this ideology is openness [35, 36, 87]. The dedicated commitment of the ROS community developers toward open-source products is well-known. Some community members support and value this norm, while others place efficiency and software quality over the need to use only FOSS tools. In one of the

community events (ROSCon 2017), members discussed Slack as an online communication tool adopted by a group of developers in the community for discussions and collaboration. Several community members refused to use it because it was not open source, while others were happy to continue using it. One member got emotional when the item came up for discussion and asserted, "I refuse to use it. It is not open source!" (Participant, ROS Community, Paper A). Another community member joined the opposition: "It is disappointing to see some people using a closed source product, but I refuse to use it." (Participant, ROS Community, Paper A). There was an awkward silence before the discussion advanced to another subject. Apparently, not all community members rank openness equally high. Some have a relaxed and pragmatic attitude toward adopting closed source infrastructure and tooling if it results in higher quality software products created more efficiently.

Another aspect of this ideology is enjoyment. While not enjoyable, non-programming tasks are executed as part of the commitment to the community. Thanks to the intrinsic enjoyment, other tasks in the software development process, such as coding, provide motivation to participate in the community and deliver high-quality code. The self-determination theory (SDT) [95, 108] explains that enjoyment comes as the tasks allow individuals to feel competent, provide a sense of belonging, and independence. Freedom of choice, the presence of challenges, and the ability to overcome challenges stimulate intrinsic motivation. Adding fun to non-coding tasks is a way, according to SDT, to overcome the challenge of necessary, but mundane tasks.

Somewhat surprisingly, I found that quality is not a high priority in the community. Innovation and functional depth and breadth are the priorities of the ROS community, and quality practices and continuous improvement are not a priority. I learned later during that other FOSS communities invest their passion in producing high quality code [50]; while ROS prefers to shift its passion to advance the cause of robotics. This difference is historic and due to the post-war cultural entanglement of new technologies, and robotics in particular. Robots became exemplars of a technological headway. This

cultural background also perpetuates the ROS community views of what their priorities should look like and how it should behave.

The third force influencing quality in the ROS community is meritocracy. Fame and reputation are the rewards for superior technical knowledge, but no reward is given to those who perform testing or documentation tasks. The culture of ROS is features first and quality later.

The absence of a working sustainability strategy constrains quality assurance activities, as well. Finding a balance between quality and stimulating growth is a challenge for the community as well. There are few initiatives to attract new maintainers, and even these have been unsuccessful. Therefore, a large number of software packages end up being orphans (not being maintained and/or have no maintainer assigned to them). Documentation is not updated either. Sustainability is healthy from an economic, environmental, and social viewpoint, and one part of sustainability is resilience. Without a strategy for improving sustainability, communities often vanish. However, sustainability is difficult to achieve, and a project that can attract and retain resources can produce more creative contributions.

The complexity of robotics systems adds additional challenges to the implementation of quality assurance. Robots are complex, combining control, AI, concurrency and mobility. These disciplines require additional quality measures so that robots can operate without defects.

The ROS community demonstrates an ability to adapt, in that it prefers an organic development of practices, rather than following prescribed practices. Trial-and-error strategy is used to develop the practices of the ROS community's quality. These are often managed informally through democratically agreed upon measures. These quality assurance measures are informal, rather than top-down rules and regulations. The community members reflect, deliberate, and vote on practices before adopting them.

Finally, the ROS community favors quality assurance measures that are

easy to use. They should be effective and efficient, and they cannot delay or constrain the developer's focus on innovation. They cannot constrain the creativity and participation of the developers.

4.5 Contributions

The results of this study demonstrate that quality cannot be investigated and defined in isolation of its social and organizational environment. Therefore, many of the traditional strategies used to increase quality, monetary incentives, training, and sharing of best practices, for instance, have partial effect. I am not suggesting that we should abandon tools; however, they should be used to support rules-based quality measures, not as the underpinnings of a true culture of quality.

Not all FOSS communities are equal. I learned in the course of my PhD project that each community is influenced by external and internal strands. For example, Linus Torvalds style of leadership has influenced the governance of the code change process (**Paper E**) in the Linux Kernel community. It reflects his personal values like trustworthiness and appraising relationships. Other communities have been strongly influenced by the unarticulated tenets of the hacker ethos, the desire to maintain pluralism, passion for quality and coding, care for the community, etc.

The learning from this study has also influenced my approach to implementing interventions in the ROS community. I had to be sensible and considerate to the particularities of the ROS community (**Paper D**). Instead of applying an action research method in a literal manner or sense, I decided to tailor participatory action research to fit the ROS community particularities and collaborative style of work.

5

Personal Motivation for Quality (Paper B)

5.1 Summary

This study collected data from five FOSS communities and analyzed the data using qualitative methods to identify factors affecting the effectiveness of code review. Qualities of individuals in FOSS communities were identified to contribute to the success of code review in FOSS communities. These qualities are passion for the project, caring about the project, concern with reputation and status in the community, and learning from reviewing other's code. Intrinsic and extrinsic motivators were also identified to elevate efficiency of code review. From this study, we were able to conclude that quality is achieved by other means apart from tools and processes. These social attributes when invested in the task of code review lead to a higher code quality.

5.2 Motivation

Code review is an established practice, in FOSS communities, that ensures quality of source code, lowers bug frequency, and enforces standards. Code review has evolved to a formal technique that gives immediate feedback, and has several forms, including pair programming, informal walkthrough, and mandatory approvals before code is merged. Code review effectiveness

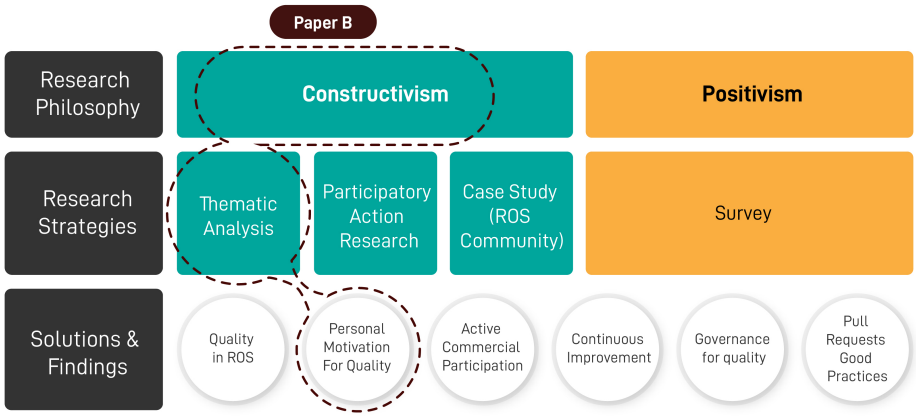

Figure 5.1: Paper B Research Methods

depends on level of participation in code review, size of changes made, and the reviewer experience and expertise.

FOSS is a gift economy, centered around reciprocity, sharing success, displaying status and reputation, and continuous learning in lieu of monetary rewards. Given this distinction, why does code review work for FOSS communities? Understanding the reasons and the motives would allow us to disclose the other facets of quality.

5.3 Methods

For this paper, I investigated five FOSS community in this study; namely, ROS, a middleware for robotics; Apache Allura, a hosting platform; The Comprehensive Knowledge Archive Network (CKAN), a storage and distribution platform for data; FOSSASIA, a community producing software applications, hardware, and design; and the Linux Kernel, the most popular operating

system kernel.

The method used for this analysis was a qualitative method (Figure 5.1). The data consisted of semi-structured interviews with 21 participants. The data collected was analyzed using open coding [75,93] to organize the findings.

5.4 Results

Rejections are common in code review, and, in some instances, the default initial position of code review is a rejection of code; therefore, a community implementing code review should create an environment where rejections are common, accepted, and normal. Mentoring and training of contributors improves acceptance of this negative feedback. FOSS contributors develop a remarkable ability to handle rejections. For example, this participant stated, *"my code was rejected multiple times. I learned not take it personally; instead I learned from the rejections how to be a better programmer"*(Participant 1, Allura Community, Paper B).

> ***Rejections.*** *Contributors are subject to frequent rejections in code review. Communities neither reduce nor eliminate the negative feedback, as they believe it is core to the practice.*

Code review is an iterative improvement cycle in the studied communities. Each iteration strives for the best code quality. These cycles of improvement turn negative feedback into a positive opportunity for learning and growth. Despite the harshness of the process, our participants appear enthusiastic to learn, leverage the process to build reputation and excel.

> ***Learning.*** *The iterative improvement cycle in code review turns negative feedback into a positive opportunity for learning and technical-growth by contributors. Receiving feedback may even become a reason for participation.*

The success of code review can be also attributed to a phenomenon called

hacker ethics, which is passion, caring, creativity, and joy in creating software. The ethic of passion is a trait that contribute in reaching high achievement, and creativity, but it also aids in coping with negative feedback. Passion can be categorized as obsessive passion, which is tied to a person's self-esteem, and can be negative; or harmonious passion, which is more positive. The passion of hackers is harmonious, and it can lead to better concentration, flow, and persistence. Passion is a strong motivator in FOSS communities. Code review is conducted with passion. Contributors invest their passion in the review task; this leads to higher achievement and, consequently, to better code quality. For example, this participant stated, *"if I don't review code at least three times a week, I don't feel good about it. In open source, our passion drives us to be exceptionally good and proficient at what we do"*(Participant 9, FOSSASIA Community, Paper B).

> ***Passion.*** *The ethic of passion motivates FOSS contributors. Consequently, they dedicate effort to code review, deliver high quality, and are more resilient to rejection.*

Hackers have been described as prizing self-determination, technological expertise and freedom; but they also exhibit care values. Caring for the community and the project is a part of the hacker ethic. There is a positive correlation between caring and performance. Caring about quality is a demonstration of care toward the community. When this care is exercised on the task of code review, the performance of the review is high. This enhances the quality of the review and assists in producing high quality code. This quote from a participant interview illustrates this findings, *"I personally care about my community. When I review code, it is important to me that it is of high quality"*(Participant 13, FOSSASIA Community, Paper B).

> ***Caring.*** *The ethic of care drives our subjects. They use the gate of code review to exercise care for quality. Care also helps them to control the negative feedback.*

Altruism and enjoyment are key intrinsic motivators to participate in

FOSS communities. These qualities are put into execution during code review which leads to higher quality of the review. Our participants described the experience as "good feeling" and "it feels nice."

> ***Intrinsic Motivation.*** *Altruism, and enjoyment are key intrinsic motivators for our subjects. Open source reviewers are effectively volunteers (even if paid) and can choose review tasks following intrinsic interests.*

Extrinsic motivators include reciprocity, participation in successful projects, status in the community, reputation enhancement, and learning opportunities. Reputation is the most pronounced motivator we found in this study. It drives contributors to participate and excel in code review in order to build up reputation in the community. This desire to perform exceptionally well produces high quality outcome, i.e. good reviews and consequently higher quality of code. For example, this participant stated, *"It is important to have a reputation in the community. This is achieved through the respect expressed by the community in the form of feedback to its members. Basically, you receive good feedback when your code is of excellent quality and your contribution impresses the reviewers"*(Participant 21, Linux Community, Paper B).

> ***Extrinsic Motivation.*** *An established reciprocal gift culture, sharing in the fame of success, reputation, public visibility of status for employers, learning opportunities, and punishment for not performing ultimately the best are the key extrinsic motivators behind work and code review of our subjects. These are all non-monetary motivators that can be used to improve code review.*

5.5 Contributions

I identified a set of social qualities (e.g. Passion, care and extrinsic motives) that lead to excellence in performing the task of code review. These social traits enhance the quality of the process (i.e. the execution) and the outcome

(i.e. high performance and better code quality). These findings show that quality has a social facet that should be acknowledged and nurtured in software development environments. It is also an indication that quality can be achieved with social traits in conjunction with tools, standards and the adherence to software engineering principles.

6

Affiliated Participation (Paper C)

6.1 Summary

This study was designed to discover the participation models prevalent in affiliated participation in FOSS communities (the participation of industry engineers in open source communities as part of their jobs) and the barriers to active participation by companies. The data was collected from semi-structured interviews and analyzed qualitatively. The research questions that are the foundation of this study are what participation models are used in affiliated participation efforts, and what barriers exist for employees to contribute to FOSS.

6.2 Motivation

Some claim that it might be impossible to find an organization today that does not benefit in some way from open source software [11]. Some companies, like Intel, IBM, and Samsung, have entire programs devoted to contributing to open source communities. Other companies become consumers of open source almost accidentally when the software is brought in by system administrators or developers. The existing literature proposes various participation models. For example, Hecker describes two main business models for FOSS, that of *support seller* and *loss leader* [48]. Fitzgerald and Kenny identified

four adoption models, *value-adding service*, *market creation*, *leveraging community development*, and *leveraging the FOSS brand* [37]. In addition, FOSS communities and companies were described as having a parasitic relationship, where the company is unconcerned with its effect on FOSS; a symbiotic relationship (mutually beneficial relationships, in which both the firm and the community gain advantage), where the relationship is mutually beneficial; and a commensalistic relationship (relationships between the two entities where one party, the firm, benefits from the other without affecting negatively the FOSS community), where one entity benefits but does not harm the other [23, 68].

However, all these works do not explain how this participation affects the software development process and particularly achieving quality. I observe a passive attitude from companies toward contributing to the software development and its artifacts. Although, over the last three years, I observed some significant change in the ROS 2 project, ROS 1 has suffered significantly from this attitude. The passive behavior makes sustaining quality in ROS a real challenge. Many artifacts (e.g. documentation, tutorials, etc.) are quickly getting out of date and quality assurance practices are easily neglected. Given these circumstances and the state it creates for quality in the community, I set to investigate the reasons that makes commercial institutions passive in FOSS communities.

6.3 Methods

Epistemology is concerned with providing a philosophical grounding for deciding what kind of knowledge are possible and how we ensure it is adequate and legitimate [71]. The epistemological stance of this study is constructivism (6.1) [29]. The aim of constructivist research is to understand particular situations or phenomena in-depth. It suggests gathering rich data from which conclusions can be formed. To unveil the unknowns about passive

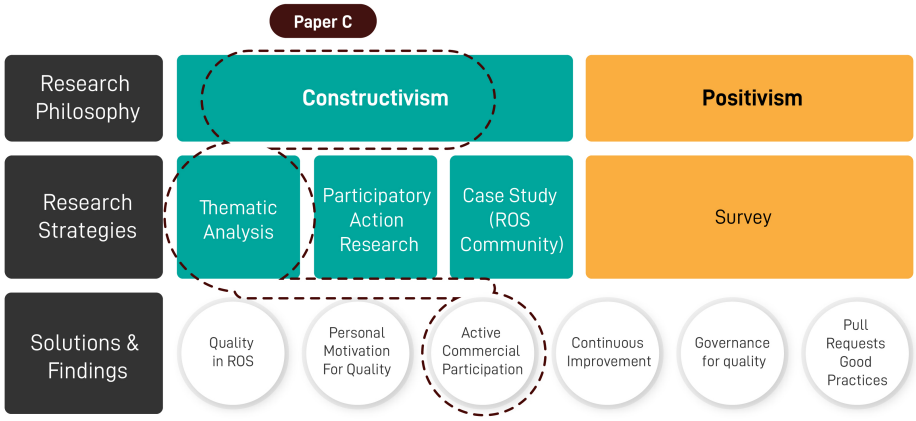

Figure 6.1: Paper C Research Methods

participation, I needed insights to people experiences, decisions and ways of thinking. This is achievable via semi-structured interviews.

Figure 6.1 shows the method employed to gather and analyze data for this study. I used qualitative methods to investigate these questions based on data collected in semi-structured interviews [46, 78] and participatory observations [77]. I interviewed 21 members of the ROS and Linux communities. All my interviewees were affiliated ROS or Linux Kernel contributors. Their professional experiences ranged from five years to 30 years and they had various responsibilities in their companies (e.g. Director, project manager and developers). The observations were conducted in the ROS community. I used thematic coding [75, 93] to analyze the data that was collected.

6.4 Results

Some of the organizations I researched had decided to benefit from FOSS in an inbound-only manner without contributing back to the community. I

referred to this behavior as **passive participation**. Passive participation is desired by management, but often not by engineers involved with the community. Management's reasons for not wishing to participate actively include intellectual property concerns, company image concerns, unfamiliarity with the FOSS cost/benefit model, and lack of clear participation policies.

This passive behaviour is explained by this participant, who stated, *"it is convenient for these companies not to contribute. There are many reasons, but the most important one is the cost to contribute is high. Producing code of the required quality is not that easy! Plus going through cycles of reviews is lentghy"* (Participant 18, Linux Community, Paper C).

Subjects in the study indicated that it is possible to develop an active FOSS participation strategy over time. Some organizations use a **latent model**, where the release of internally developed features is delayed until an economic gain has been guaranteed. This latent model can benefit both the company and the FOSS community. It allows developers to become embedded in the community and influence the direction of the project, as well as push the organization toward sharing more. Selective revealing is also implemented where the commodity parts of the project are contributed and differentiated components are kept closed. Latent participants neutralize risks of disclosing differentiated IP, yet benefit from better embedding into the community.

Enterprises can engage in active participation in an Open Source communities. It is a strategic decision to combine in-bound and outbound engagement with the community. Active participants seemed enthusiastic and devout about it. It is shown in this participant statement. *"We see ourselves an integral part of the community. All the code we produced is published for review. We adhere to the community standards and passion for quality and we believe in a meritocratic system of rewarding achievements internally .... We wouldn't exist without the community; our business model is built around it."*(Participant 21, Linux Community, Paper C).

I identified six barriers that inhibit affiliated participation. Poor understanding of FOSS can lead senior management to resist active contributions to the community, and subjects indicated that the entire company must be committed to active participation. Other barriers identified were company image concerns, intellectual property protections, undefined processes, high cost of participation, and unfamiliarity with open source.

Passive and latent participants object to active participation because of concerns over intellectual property. Active engagement is expensive, both financially and psychologically. For example, the Linux community has a high barrier of entry, and costs include preparing the code of high quality, accepting rejections, and going through multiple review cycles before a contribution can be accepted.

6.5 Contributions

Achieving quality is reactive to the environment where quality is produced. Increasing passive participation in the ROS community impacted the sustainability of quality in this community. When commercial participation in FOSS communities increases, understanding the participatory behavior of these organizations and how it affects quality becomes increasingly important. The attitude taken by these organizations constitutes the link between achieving quality in FOSS and commercial participation. Using a community-developed free software or resources and expecting that somebody else will step in to contribute (e.g. fix bugs, update documentation, etc.) is selfish and not sustainable. Quality is a believe and a shared-interest in FOSS communities. Passive participation is not in line with achieving common and group interest.

This study draws lessons for communities leaders (or managers) and senior management of companies participating in FOSS communities. These lessons are listed below:

- Community leaders should institute and maintain a dialogue with companies senior management. The purpose of such dialogue is to advocate for active participation. Active participation does not always mean releasing critical and competitive features. It can be contributions to fix bugs, update documentation or financial contributions to the foundation supporting the community.

- Senior management should investigate the community image and reputation prior to participating. It is a relationship and not a simple download of the software. Hence, management should invest some time in knowing the community they are about to adopt its code.

- If the protection of intellectual property is the issue, then there are various means to actively participate. As stated earlier, fixing bugs, updating documentation and mentoring newcomers are contributions.

- Participating in FOSS does not simply imply downloading code. Companies need to put in place a process and policies to guide their engineers in using and participating in FOSS.

- The cost of participation is primarily generated by the FOSS community itself. FOSS communities should reflect in their processes and make them more fair and friendly (see **Paper F**).

7

Continuous Improvement (Paper D)

7.1 Summary

The purpose of this study was to positively influence the ROS community quality practices by implementing a set of interventions. I designed PAR4FOSS, a framework for implementing change specifically for FOSS communities. This framework is tailored for FOSS particularities to ensure a social and cultural fit. The framework has been instantiated for the ROS community, then implemented. The experiment showed that when the motivation for quality combined with the availability of resources (i.e. contributors to work on quality initiatives) coincide, then quality interventions are implemented.

7.2 Motivation

Many FOSS projects have existed for long enough to accumulate changes with the software quality (see Chapter 4). Addressing quality changes requires introducing change to the community. I designed and executed a tailored participatory action research (PAR) method to determine the ability of a FOSS community to embrace a quality improvement initiatives.

7.3 Methods

Participatory Action Research (PAR) is motivated by both practical concerns and concerns of equity. Most participatory research focuses on "knowledge for action", or a bottom up approach with the focus on locally defined priorities and local viewpoints that arises as involving local people as participants in the research and planning. PAR enhances effectiveness and saves time. A key element of PAR is the attitudes of researchers who determine how and for whom research is conceptualized and conducted. PAR enables local people to seek their own solutions according to their priorities [60].

PAR is reflexive, flexible and iterative, with a key strength of exploring local knowledge and perceptions. Researchers become learners, as well as facilitators and catalysts for action. The most important element of action research is not the theories, but who defines the research problems, and who generates, analyses, represents, owns, and acts on the information which is sought. Participatory Action Research analysis focuses attention on the central issues of power and control. The control lies, not in the researcher of the participants, but is shared, in the form of a zig-zag pathway with greater or less participation at various stages [60].

The strengths of action research are its capacity to provide robust real world application and testing of findings, research interventions that are specific and targeted, and result in opportunities for stakeholder engagement. More successful outcomes occur depending on the degree of negotiated access to the project setting, clear role outlines, commitment of time building relationships, sensitivity toward insiders, and the scope of the project allowing for flexibility [60].

The proposed method is a tailored version of PAR to fit FOSS. It was evaluated in the ROS community as a case study (Figure 7.1). Participatory action research focuses on research that enables action through reflective cycles with both participants and researchers. It is more democratic, builds capacity,

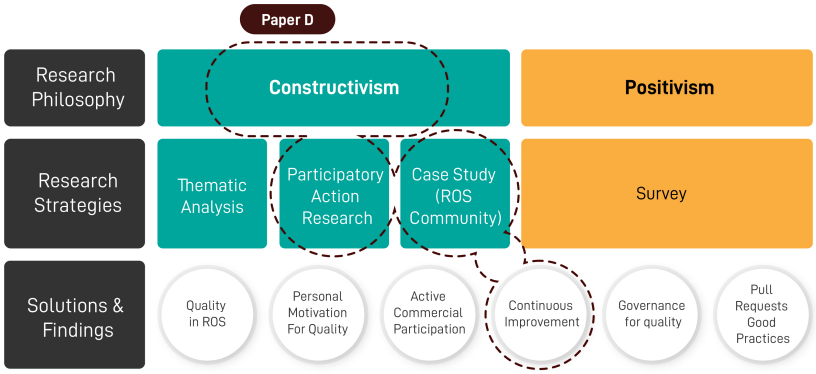

Figure 7.1: Paper D research Methods

and encourages self-determination. The PAR process begins with reflection, followed by action by researchers in concert with participants' observations obtained with informal discussions, and then the process cycles back through these steps. PAR results in more self-reported self-confidence, self-awareness. PAR can follow either a rigorous structure and a fluid structure [5]. I needed to tailor the current guidelines of PAR to fit the cultural and social particularities of ROS.

7.4 Results

I designed a participatory action research framework for FOSS communities, called PAR4FOSS (Figure 7.2). PAR4FOSS is a new framework applicable to FOSS communities that has three components; namely, interventions design, democratization, and execution. It is an iterative process where transparency and open decisions are implemented by self-management and collaboration. The interventions are obtained from community requirements, similar com-

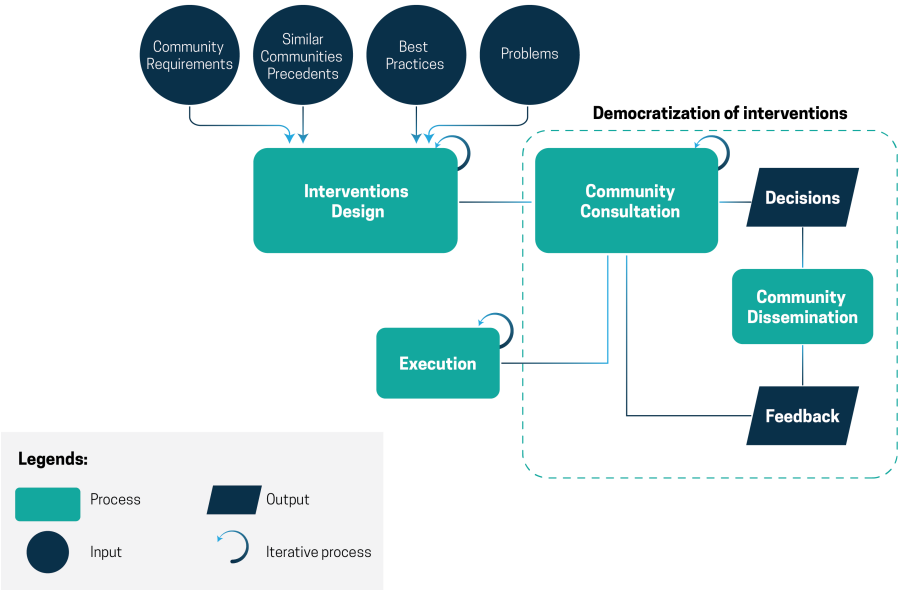

Figure 7.2: PAR4FOSS Framework

munities' precedents, best practices, and problems. The PAR4FOSS has two phases, pre-interventions and interventions phases. Four iterations were followed in this case, following the PAR4FOSS method, to collaboratively promote quality assurance best practices and construct a culture where quality is part of the fabric of the community.

The interventions are designed and enhanced iteratively, obtained from community requirements, similar community precedents, best practices, and problems. The community requirements exist as a legitimate need to improve a process, and other communities are examined to see what they did to successfully ingrain a culture of quality. The best practices are industry practices that enhance a situation, and the problems of the community are related to the scope of the change. The problem is defined as the difference between what is happening and what should be happening. The democratization of interventions is to legitimize the interventions, and the interventions come from community consultation and dissemination. The

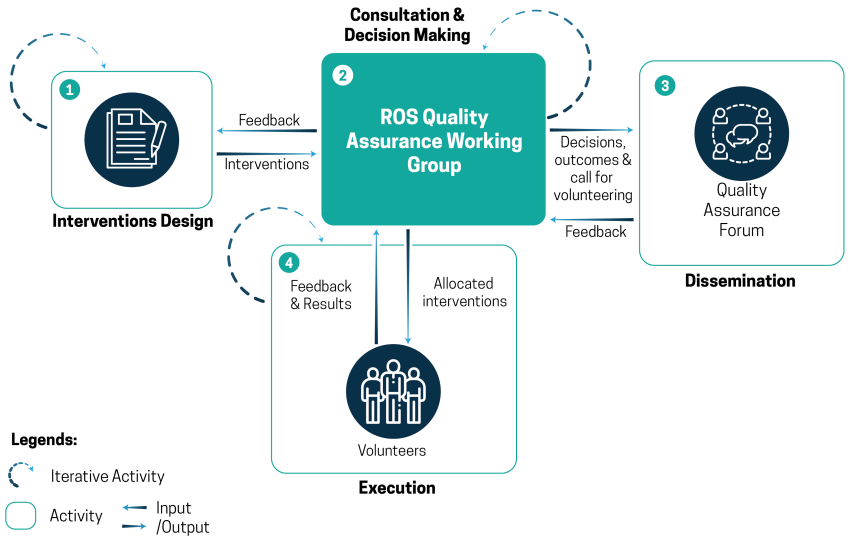

Figure 7.3: PAR4FOSS Instantiation for ROS

execution is the implementation of the interventions.

Figure 7.3 is the PAR4ROS instance deployed for ROS. In the application of PAR4FOSS toward the ROS community, I created a working group called the ROS Quality Assurance Working Group. This group became the platform for the action. In the first phase of the project, the community participated in identifying and analyzing the problems, and the second phase implemented iterations of planning, acting, observing, and reflecting. This research intervention creates change by influencing individuals' knowledge, attitudes, and beliefs, by increasing quality assurance (QA) infrastructure and tools, and by creating supportive environments for QA practices. The group met monthly and their discussions were summarized and publicized in the community quality assurance forum. Feedback was received from the community and then communicated back to the group.

We (The QA working group and I) created an impact in the community and we successfully implemented three interventions in ROS 1 and two inter-

ventions in ROS 2 using volunteers from the community. Given that ROS has lack of motivations for quality **(Paper A)**, this is a considerably good achievement and shows the method can deliver. Whether the interventions have created a positive outcome that is an entirely different question. Assessing the impact of interventions is a test for the assumptions the interventions make, i.e. if the change occurs then a positive impact will consequently follow. In other words, when we test an interventions we test the assumption that was made at the time of its design, which is making positive impact not a test of the method used to deliver the intervention.

7.5 Contributions

The novelty of this research is the ability to introduce change to a FOSS community through the PAR4FOSS method. The lessons learned were that competing priorities hinder the motivation to participate, that volunteers can be unreliable, that most volunteers in the community have other jobs and commitments, and that the method should be flexible and adaptive. I also learned that securing the participation of resources with the right skills is a challenge.

8

Governance for Quality (Paper E)

8.1 Summary

This chapter summarizes a qualitative study (**Paper E**) into the decision-making process of pull request (PR) acceptance in FOSS communities. Decision-making is the process of identifying and choosing contributions based on the values, preferences and beliefs of the community. The success and sustainability of FOSS communities depend on ongoing contributions, therefore, the factors and values that drive the community decisions are important to understand. This qualitative study sought to determine what factors affect pull request (PR) acceptance and the principles behind pull request evaluations. I interviewed 30 participants from five communities to collect data for this study. The data was then coded and themes relevant to the PR process emerged. I found that acceptance depends not only on technical, but also social, and strategic factors. I identified and described three types of pull requests governance; namely, protective, equitable, and lenient, and determined how each style motivates contributors and maintains software quality in FOSS.

8.2 Motivation

Contributors in FOSS communities submit pull requests for code changes, such as a bug fix or new feature. The pull requests are reviewed for appropriateness and quality, but the process is different in different communities. Few pull requests (PR) are rejected because of technical reasons, but social factors play into rejections as well [40]. The PR evaluation process is critical to a growth of the functional depth and breadth of FOSS product. Understanding the governance model and its correlation with PR evaluations may help understanding how quality is sustained across different communities..

8.3 Methods

A decision to merge or reject a PR is a complex decision that is a result of many factors. Therefore, in evaluating the processes that take place in the assessment of PRs in open source communities, an examination of both human and social aspects is critical. Consequently, I chose a qualitative method (Figure 8.1) of research that is suitable for revealing and gaining insights into a participants' experiences and perspectives, resulting in rich data. I interviewed 30 contributors and maintainers from five communities with experiences ranging from two years to 30 years. The interviewees are geographically distributed across Asia, Europe and North America. We analyzed the interview data using thematic coding [75, 93] which allowed us to reveal complex mechanisms, behavior and rational. For example, the Coala community believes that rejections are "rude" and strategically wrong. Rejections deter contributors and undervalue their enthusiasm. Instead, the enthusiasm of submitting a PR is leveraged; when the contribution doesn't meet the community standards, the contributor is mentored to bring her code to the quality required. It is believed that this behavior toward the contributor will convert him or her to a productive member of the community. In contrast, the Linux Kernel community believe that rejections are a defense

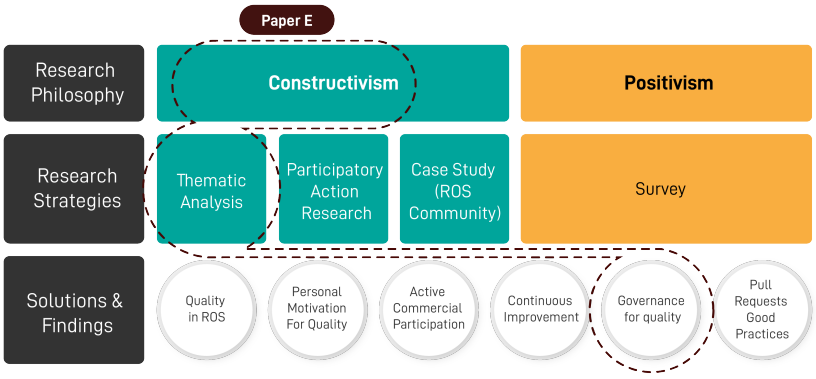

Figure 8.1: Paper E research Methods

mechanism against bad contribution. The community has a profound belief that the fair of rejection drives contributors to excel in order to impress the gatekeeper.

8.4 Results

Each of the examined communities shows a consistency in governance that create a culture of excellence where quality is supreme. The protective and lenient styles of governance show that the person matters as well as the code quality. The equitable style focuses primarily on code quality and meeting community standards.

The protective style of governance is a defensive style where the project leader and his subordinates have absolute power over merges. This style is referred to as the "no by default." Maintainers reject code change submissions without the necessary due diligence, based on the perception that the contributor is unreliable or untrustworthy. This finding is illustrated in this

statement of a Linux Kernel engineer, who stated, *"On some parts of the kernel building trust is essential, and there is a clear social entry barrier. It has some downsides for beginners. Yet it's understandable, as changes in the kernel always come with some kind of maintenance overhead, and maintainers want people that have proven to take ownership of their contributions ... However, once a patch is considered, then it goes through thorough vetting."* (Participant 24, Linux Community, Paper E)

> ***Protective*** is the PR governance style that relies on trust, relationship building and the contributor's reliability.

The equitable style is based on fairness, where acceptance is a balanced and grounded technically. Rejections are not taken lightly, but based on justified technical argument. Participant 6 explains, *"There are principles for evaluating PRs and we religiously obey by them. We follow principles to be the most important, and will usually reject a PR if it doesn't hold up to these principles."* (Participant 6, Odoo Community, Paper E)

> ***Equitable*** is the PR governance styles that values fairness and rigorous application of community principles.

The lenient style is positive and welcoming, tolerating more errors, but still quality is not compromised. The lenient style sees contributions as assets and believe all contributions should be welcomed enthusiastically. Mentoring is used to develop newcomers' expertise and elevate the quality of contributions to the required community standards. This perspective was explained by participant 18. He stated, *"Rejections kill motivation and it's a rude thing. We instead steer the contribution to a positive direction, by making it better and get it merged."* (Participant 18, DuckDuckGo Community, Paper E)

> ***Lenient*** is the PR governance style that reduces social barriers and assumes that every contribution can be elevated to a mergeable state.

So how these governance styles contribute to achieving quality in FOSS? All three PR governance styles help to reduce the problems with poor code. Consistency and governance help in the creation of a culture of excellence. One participant said, *"The quality is the main driver that drives our decision to either accept or reject a PR. The processes are there to support and control the decision-making."* (Participant 2, FOSSASIA Community, Paper E). The governance of PRs contributes to the sustainability of software quality in FOSS. This participant further explains, *"first, reliability of the code. Open source is ever changing, people come and go. High quality code and the ability to read the code and understand it is critical."* (Participant 2, FOSSASIA Community, Paper E). This view was repeated concerning all PR governance styles. One participant supporting the lenient type of governance said, *"We keep contribution's code quality in check, but at the same time, we are trying to be lenient towards contributors to really help them out to get the codes to the level where their submission can be merged."* (Participant 29, Coala Community, Paper E)

Once a PR is considered for a review, a set of software engineering principles are applied to assess its eligibility to be merged. The quality of contributions is assessed using seven principles. These principles are atomicity of the contribution, maintainability of the code, avoiding technical debt, passing peer code review, compliance with community best practices, documentation, and passing tests.

8.5 Contributions

This study identified three governance styles; namely, protective, equitable, and lenient. The study also found that the PR governance is a quality measure

to ensure high-quality contributions. All three types of PR governance deliver quality outcomes that are a combination of the creation and the ongoing use of systems that reinforce the consistency and repeated efforts of the processes. This mean that disciplinary measures (i.e. rules and control) are put in place to ensure ongoing delivery of high quality code. Each of the models, however, works toward quality in a different ways. The communities are aware of the model they follow, and appears to be an important part of their identity.

9

Pull Requests Good Practices (Paper F)

9.1 Summary

Because of the need to retain contributors, a positive experience is important for contributors. Delivering positive contributor experiences means the community recognize the contributor's effort invested in developing the code, and more contributors means evolution of the software. Positive contributor experience help build better relationships between the community and the contributors and create a welcoming and positive climate. This study used a qualitative method to analyze data from surveys on what constitutes a fair and an unfair pull request evaluation. I identified seven good practices (engagement, communication, appropriateness, simplicity, compliance, support, and decision) that make the experience perceived as positive by the contributors and can enhance the experience for all stakeholders (contributor, community and maintainer).

9.2 Motivation

GitHub receives 200 million pull requests (PR) from over 31 million developers in one year, most of these go through open evaluation in public. In this process, some contributors experience positive circumstances, but others do not. I set out to find out what makes an evaluation fair and what behaviors are

seen contributors as unfair. A rejection of a PR often results in demotivating contributors who may cease contributing to the community [107]. While recruitment and maintenance of new contributors remain critical for most FOSS communities.

Teams that are geographically distributed experience more miscommunications and misunderstandings, and they have more difficulty sharing information and feedback. According to Hinds and McGrath, members of a team can be categorized as constructive, passive, or aggressive [51]. Constructive members are balanced and show cooperation, creativity, and a free exchange of information and respect. The passive style places more emphasis on fulfillment of relationship and reputation goals, maintaining harmony, and limited information sharing. The aggressive style places emphasis on personal achievement with personal ambitions placed above concern for the group. Constructive groups produce superior solutions, compared to passive and aggressive members. [51, 90]

When altruistic reasons for volunteering are not satisfied, volunteers are less likely to remain with the organization [73]. Volunteer retention occurs when motives for joining are satisfied, and when alignment with goals and values exists. Social support increase retention of volunteers [117]. To retain participants, community developers need to make communication channels responsive to newcomers and make sure the interactions are positive [14]. The probability of a newcomer becoming an long term contributor was found to be associated with the person's extent of involvement and interactions with her environment [119]. In the ROS community, I observe that challenges in sustaining the community (i.e. mainly retention of contributors) affect negatively the overall quality of the processes in place. Given the pull requests process is the main entry point of newcomers, I decided to investigate what makes the PR evaluation process positive and supportive for contributors, as to encourage them to remain loyal to the community.

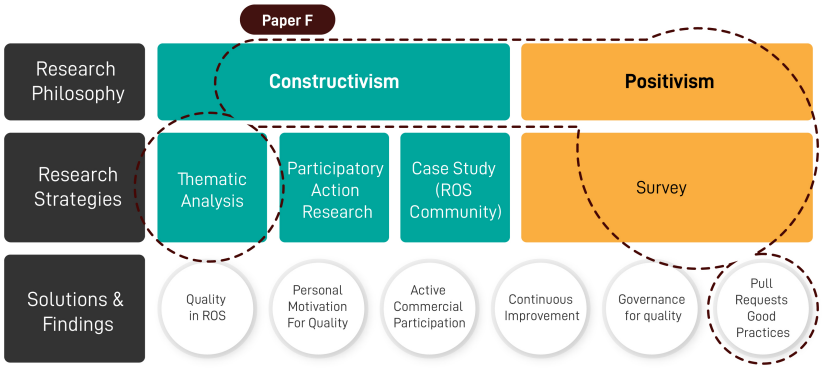

Figure 9.1: Paper F research Methods

9.3 Methods

As shown in Figure 9.1, this study's method used a survey to collect qualitative data (links to fairly and unfairly assessed PRs and the participants' justification for their choices). I selected eight communities for this study, including ROS, FOSSASIA. Coala, Plone, Apache Spark project, OpenSUSE, Linux Kernel, and OpenGenus. The respondents were invited to participate through emails from GitHub. Some participants were found on community forums and mailing lists. The respondents were asked to find one fair and one unfair pull request evaluation, and to give a justification of their choices. I received 48 cases of fairly assessed PRs and 10 unfairly assessed ones.

The dialogue around the PRs in Github was analyzed using a thematic categorization framework, and patterns were identified [75, 93]. The patterns became themes, and the themes were labeled, resulting in the seven good practices that make up the findings of the study. This kind of data and analysis primarily reveal behaviors ans statements made by the contributors and evaluators in public. It might be different than interviews data, but it

still allows to capture behaviors, believes and perceptions.

9.4 Results

I identified a set of good practices found in pull request processes that have been deemed fair. One practice is engagement, which I described as the enthusiastic and warm reception from the community. When the number of reviewers, engaged in the PR, exceeds two, the PR process is deemed fair. Engagement acknowledges the contributor for his or her effort, and it includes providing feedback, and a decision about the contribution. Lack of engagement is demotivating, and it results in a loss of contributors.

> ***Engagement.*** *Every contribution carries enthusiasm, which should be rewarded in the form of acknowledgement, feedback, and a decision. It is a good habit for maintainers and senior community members to ensure that new contributions are discussed timely, that feedback is provided, and decisions are made.*

Another practice is good communication, which refers to the tone of the interactions being positive and professional. Good communication steers the evaluation process in a positive direction. Abrupt changes of subjects in a conversation prevent it from being constructive. However, professional, constructive, and supportive discussions help the community.

> ***Communication.*** *Maintainers and reviewers should develop a habit to communicate objectively, clearly and professionally.*

Contributions must meet the needs and plans of the community's vision and roadmap. Contributions must meet the needs and plans of the community's vision and roadmap. Appropriateness is the quality of being suitable or when a contribution addresses a particular community requirement, such as fixing a bug or a desired feature.

> ***Appropriateness.*** *Contributors should develop a habit to consult the community issues list and the product vision documentation when available, to ensure that their contribution addresses a legitimate need. Communities should document their product vision and promote it towards contributors.*

FOSS communities prefer straightforward, modular changes because they are easy to review and assess. Cumbersome pull requests increase the risk of defects in the code. Contributors should develop and submit easy to understand, modular submissions.

> ***Simplicity.*** *This quality is highly appreciated in the contributions. Contributors should develop a habit of developing and submitting easy to understand and modular submissions.*

Compliance in FOSS communities is strictly enforced. Rules may apply to programming language guidelines, while other delineate conduct and communication. Contributors should make sure that their pull request is compliant with the community guidelines, or the contributor is viewed as incompetent.

> ***Compliance.*** *Contributors should ensure that their submission is compliant with community guidelines and rules. Compliance signals competence of the contributor and recognition of the competence of the community.*

The support of community members encourages a productive contribution of a significant quality that can be merged. It also enhances team cohesion, and helps elevate the quality of the submission. A lack of support is indicated by minimal feedback to contributors and by focus on errors, rather than offering help and mentorship.

> **Support.** A supportive evaluation process creates a positive atmosphere and gives a good experience to contributors. Reviewers and maintainers should develop a habit of being constructive and collegial.

A logical, fair decision that is based on technical merit is also a good practice for communities.

> ***Decision.*** *Decisions should be based on technical grounds. Maintainers should develop a habit of communicating the decision rationale clearly, in technical language.*

Communities should encourage reviewers to be warm, welcoming, and supportive. Contributors should be assertive and ask for help when needed. An evaluation based on technical merit should always be the focus of the community.

9.5 Contributions

Assuring quality requires an in-depth knowledge of the product, the community, the history, and the processes. A contributor who is thoroughly familiar with and integrated with the community will produce high-quality contributions. Such contributor is difficult to replace. Any new contributor will need time to fully integrate with the community, learn the community's goals, understand the product's current state, and absorb the history of the quality process up.

The need to retain your contributors is as important for quality as with any other technical function, and can greatly impact the quality, effectiveness and speed of software quality assurance efforts. I identified seven good practices of PR evaluation with the purpose to enhance the overall experience for all concerned stakeholders. To sustain quality in FOSS, communities should encourage ongoing flow of contributions and reviewers.

High rates of attrition affect how well a contributor can learn the product, earn trust, gain efficiencies, and deliver the most valuable contribution possible. There is a positive relationship between high employee retention and the quality, effectiveness and speed of software quality assurance efforts.

10

Governing Pull Requests in FOSS (Quantitative Study)

10.1 Introduction

In this chapter, I will present an extension of **Paper E**. In **Paper E** and chapter 8, I investigated qualitatively how five FOSS communities (i.e. FOSSASIA, DuckDuckGo, Linux Kernel, Odoo and Coala) govern their PRs process. I identified three styles of governance: (1) protective, (2) equitable and (3) lenient. The aim of a qualitative study was not to prove generalizable conclusions but to provide a rich, contextualized understanding of a given phenomenon through the intensive study of particular cases [89]. To draw broader inferences, I conducted an additional quantitative study based on a survey described here (see Appendix G for the questions). I expanded the sample to cover a total of 15 communities and surveyed N=387 respondents. The data shows that acceptance of contributions depends not only on technical basis, but to a great extent also on social and human aspects.

In FOSS communities, changes to the code base are made through the pull request (PR) process, where the code is vetted for quality and conformance with the community standards. The process is packed with social practices, display of various human behavior, and decision-making. The decision making can be either conscious (active) or unconscious (without a strategy). According to Gousious, et al., only 13% of pull requests are rejected due to technical

reasons [40] and social challenges are the toughest challenges encountered by contributors [41]. This is an influencing reason to investigate the rules and control mechanisms put in place by FOSS communities to decide on the fate of PRs.

I wanted to statistically validate the following hypotheses:

Hypothesis 1. The Linux Kernel community adopts a protective style of governance for its code change process.

Hypothesis 2. The FOSSASIA community adopts an equitable style of governance for its pull request process.

Hypothesis 3. The Odoo community adopts an equitable style of governance for its pull request process.

Hypothesis 4. The Coala community adopts a lenient style of governance for its pull request process.

Hypothesis 5. Each of the 15 FOSS communities adopts a governance style, either protective, equitable or linient, for its pull request process.

Hypothesis 6. The Coala Community is more lenient than the Linux Kernel Community

10.2 Methods

I intend to understand the process that takes place in the assessment of PRs in FOSS communities; the what and how a PR is evaluated and decisions are taken to merge or reject PRs. Given the uncharted nature of the question, I opted for a mixed-method to explore and validate the outcome.

This study is a mix of methods, qualitative (**Paper E** and chapter 8) and quantitative (**this chapter**) with equal status (QUAL $\rightarrow$ QUAN). The data

collection is sequential, the qualitative phase precedes the quantitative phase. The implementation of the quantitative phase depends on the results of the qualitative phase. A sequential exploratory design is usually conducted in two phases, with the priority given to the first phase [19]. This heightened knowledge and validity of the study should be of sufficient quality to achieve multiple validities legitimation [98]. The qualitative data collection occurred before the quantitative data because I did not know what constructs are important.

10.2.1 Phase I: Qualitative

In the qualitative phase, I collected data using interviews (30 interviews) with contributors and maintainers from five communities. The five communities also participated in the quantitative phase. For further details in this phase of the study, please refer to **Paper E and Chapter 8**.

10.2.2 Phase II: Quantitative

In the second phase, a quantitative study is conducted to test the first phase's concepts. The quantitative strand thus builds on the qualitative phase, with the purpose to generalize the results to a population [19].

Population and Sampling. Respondents were N=387 FOSS contributors and maintainers from fifteen communities. Participants who failed the validation check were excluded from further analyses. I validated the data based on the following criteria:

1. The respondent must be an active contributor or maintainer in one of the selected FOSS communities.
2. All mandatory questions have been answered with valid input.

3. Only one entry per respondent. We controlled participation per one IP address.

I received 473 responses and N=387 were valid and complete.

Participation in this study was on a voluntary basis. No compensation in any form was given to the participants. We recruited the participants by:

1. Direct contact using publicly available email in the contributor's GitHub or GitLab profile.
2. Posting invites to participate in the survey in the selected FOSS communities' forums, mailing lists and chat rooms.

Instruments and Procedure. The instruments consist of a set of multiple choices and free text questions. The survey had a total of twelve questions (the survey is available in Appendix A). The key questions relevant for this analysis are listed in Table 10.1. The respondents have been asked to answer in Likert scale.

To test the questionnaire structure and to ensure that respondents fully understood the nature of the questions being asked, I conducted a pilot survey in the ROS and Coala communities. I received feedback on the design of the survey and amended the survey accordingly. Upon the completion of the pilot, we started sending and posting survey invites. I activated a survey for a period of three months in 2019.

Data Analysis. We[1] selected Bayesian data analysis to analyze the quantitative data. Bayesian analysis, also called explicit probabilistic inference, is a direct, formal means of dealing with uncertainty in scientific

[1]The Bayesian data analysis was performed by Raúl Pardo Jimenez, a colleague at the Department of Computer Science at ITU. He also implemented the Python program

ID	Survey Questions
V27	"In general I say no to most pull requests (PR)/patches."
V28	"I don't consider a pull request/patch, unless I trust the contributor."
V29	"I don't consider a pull request/patch, unless the contributor is reliable."
V30	"I don't consider a pull request/patch, unless I have a strong relationship with the contributor."
V31	"I assess every pull request/patch in the same manner irrespective of the contributor."
V32	"I assess pull requests/patches purely on technical grounds."
V33	"I never say no to a pull request/patch. If the quality of the PR/patch is not mergeable, then I mentor the contributor to elevate his/her PR/patch to a mergeable state.".

Table 10.1: The survey questions relevant for this analysis

inference. When parameters are unknown, Bayesian analysis is a way to identify the probability distribution over parameter values. An analysis of probability using Bayesian statistical analysis relies on prior distributions which allows to insert some subjectivity into the model. When new data is collected that does not fit the prior distribution, Bayesian analysis can be used each case probability across possibilities and to revise prior beliefs. Because reasoning with a probability distribution over conclusions is difficult, we often use a simplifying abstraction, the High Probability Density interval (HPD). The HPD interval describes the range of conclusions that are most credible, typically accumulating 95% of probability mass. Working with 95% HPD limits the chance of erroneous conclusions to 5% [63].

A magnitude of effect can be identified, which is referred to as a Region Of Practical Equivalence or ROPE. The ROPE is a decision threshold that is chosen in the context of current theory and measurement precision. If the ROPE, or region of values, does not include the high density values, then the value is rejected, but if the ROPE completely includes 95% of the high density values, the value is accepted because the high density values are the

most credible values [63].

Prior to analyzing the raw data, we defined groupings (Table 10.2) of PR governance styles based on the qualitative study (**Paper E**). According to the findings of **Paper E**, FOSS communities would adopt one of these PR governance styles: *Protective*, *Equitable*, and *Lenient*.

PR Governance	**Definitions**
Protective	A community is classified as "Protective" when the response from this community is positive in at least one of these variables V28 ("I don't consider a pull request/patch, unless I trust the contributor"), V29 ("I don't consider a pull request/patch, unless the contributor is reliable."), and V30 ("I don't consider a pull request/patch, unless I have a strong relationship with the contributor."). It is possible that positive evaluations of V27 ("In general I say no to most pull requests (PR)/patches") can be included in this evaluation too
Equitable	A community is classified as "Equitable" when the response from this community is positive in at least one of these variables V31 ("I assess every pull request/patch in the same manner irrespective of the contributor."), V32 ("I assess pull requests/patches purely on technical grounds."). The equitability in community is expected to find positive answers on V31, V32 which have different tone compared to V33 ("I never say no to a pull request/patch. If the quality of the PR/patch is not mergeable, then I mentor the contributor to elevate his/her PR/patch to a mergeable state.") in question.
Lenient	A community is classified as "Lenient" when the response from this community is positive V33 ("I never say no to a pull request/patch. If the quality of the PR/patch is not mergeable, then I mentor the contributor to elevate his/her PR/patch to a mergeable state.").

Table 10.2: The PR Governance Styles as Defined in **Paper E**

This survey was designed so that participants responded based on the Likert scale as follows: 1. Strongly Agree, 2. Agree, 3. Neutral, 4. Disagree, 5. Strongly Disagree. Based on the responses of participants, the results were categorized as positive if the 95% high probability density (HPD) interval falls below 3 on a linear scale (< 3).

In order to test the hypotheses discussed above, we use Bayesian inference to estimate the underlying distributions of the responses that each FOSS community gives to each of the variables (i.e. V27 - V33). We assume that the answers to the questions are normally distributed over the possible answers, so we used a Bayesian model with normal distributions as a likelihood function. We do not use a subjective prior in our analysis. We set a neutral and flexible prior, so that we let the inference find the values of the parameters for the underlying distribution that better accommodate the answers for each community. Our subjective understanding of the problem based on the qualitative analysis is instead informing the structure of the model and the choice of the hypotheses. We used PyMC3 to perform our analyses, which is a Python package for Bayesian statistical modeling and probabilistic machine learning.

10.3 Subject Communities

I intentionally sought diversity in my selection of FOSS communities. The subjects build different products, have different participation demographics and history. For example, while FOSSASIA produces multiple software solutions advancing "social change", the Linux Kernel has become the defacto choice for high performing computers. This diversity has shown to improve the richness and the generalizability of the findings.

FOSSASIA: The FOSSASIA community shares and develop software, hardware, and knowledge. The community enables people to participate in

the "sharing society", expand knowledge, tools and opportunities, freedom of communication and expression for everyone. The community has various projects being developed, but the most successful projects are SUSI.AI and EventYaY. SUSI.AI is an artificial intelligent application that provides functionality for personal assistance, Help Desks and Chatbots. EventYaY offers features for organizers to create and manage events.

Odoo: The Odoo community develops a FOSS ERP. The community developed over 30 main applications. The community is committed to create and maintain an ERP that meets the complex needs of organizations without being complicated to use. They create software that is rich in features, integrated, and easy to upgrade. The ERP offers functionality to manage and record sales, inventory, procurement, and accounting functions. It also has a business intelligence engine with an all-in-one business suite program. Odoo claims to be the most installed suite of business software for both small and large business entities. The community has more than 1500 active members.

DuckDuckGo: DuckDuckGo is a community that designs, develops and maintains a search engine committed to privacy. It was founded in February 2008 by Gabriel Weinberg, the community grew as a search engine that does not track the user's searches nor sell information about it. By 2013, there were over 3 million users on DuckDuckGo. In 2014, DuckDuckGo was included in Safari, and it was built into Mozilla.

Linux Kernel: The Linux Kernel is a free and open source operating system, developed and maintained by the Linux Kernel community. Developed in 1991 by Linus Torvalds, Linux Kernel operates under a GNU license and now reaches a large user base around the world.

Coala: The Coala community develops and maintains a language-independent analysis toolkit written in Python. Coala is referred to as a

linting and fixing program. Coala portrays itself as a newcomers welcoming. Directions for newcomers are clear and encouraging. The community documentation outlines how to get started as a member of the community, beginning with meeting others in chat rooms and Gitter. The software was first released in July 2015, and another release was made in 2015. Six additional releases were made in 2016, and two additional versions were released in 2017.

ROS: ROS is an open-source meta-operating system that provides a flexible framework for writing robot software. ROS provides a communications infrastructure that is called a middleware. It offers anonymous message passing, recording and playback of messages, request and response remote procedures, and a distributed parameter system. In addition to middleware communication, ROS provides robot specific features, such as standard message definitions, a robot geometry library, robot description language, diagnostics, pose estimation, localization, mapping, and navigation.

Plone: Plone is a content management system built on the Zope application server. Plone is positioned as an "Enterprise CMS", and it is commonly used for intranets and as part of the web presence of large organizations. High-profile public sector users of Plone include the U.S. Federal Bureau of Investigation, Brazilian Government, United Nations, City of Bern (Switzerland), New South Wales Government (Australia), and European Environment Agency. Plone's proponents cite its security track record and its accessibility as reasons to choose Plone.

ReactJS: In 2013, Facebook released its React framework as a part of a hackathon. The framework has since enjoyed huge popularity. ReactJS and React is a JavaScript library for building user interfaces. React allows users to create interactive UIs. Declarative views make code more predictable and easier to debug. React allows users to build encapsulated components that

manage their own state, then compose them to make complex UIs.

AngularJS: AngularJS is an open source platform that makes it easy to build applications for the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

OpenGenus: The community develops tools for people with bad Internet connectivity. Its projects include Cosmos, which is an offline collection of algorithms and data sources in various programming languages; Quark, which allows offline searches, and the ability to save web pages in a browser, see history, play games offline, find images offline, and find code. Other projects are Search Engine, which allows for real time safe searches; IQ, which is a community of freelancers and entrepreneurs who post questions and discuss computer issues; and Discuss, which is a flexible place for programmers to grow and get advice from others.

OpenSUSE: OpenSUSE is a Linux distribution providing a user-friendly desktop and other features. OpenSUSE includes openQA, an automated testing service used to determine if build/release updates are good. Other products of OpenSUSE are OSEM, an event management tool; Jangouts, which is a videoconferencing tool; YaST, the installation and configuration tool; and Kiwi, an application for making a wide variety of image sets available for Linux supported hardware.

Apache: Apache software develops and incubates hundreds of projects through its merit-based process, known as "the Apache way." The Apache community currently has 7000 code committers across six continents. It is based on the following beliefs of putting community before code, those who

do the work make the decisions, if it did not happen on the mailing list, it did not happen.

NodeJS: Node.js is an open source and cross-platform JavaScript runtime environment that can work as a tool for many types of projects. Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. A Node.js app is run in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

Mozilla: Mozilla is a community developing a range of Web and mobile technologies. It was founded on the legacy of Netscape. Mozilla Firefox is a browser for a web experience that is based on new standardization that includes privacy and marketing advantages. Mozilla is built on a mission of openness, innovation, and opportunity.

JQuery: The JQuery community designs and plans the future of the JQuery UI library. Being committed to maintain an open, transparent community, the community members work as a team to develop, design, and maintain widgets, animations, and class names that can be used as themes or styles. The community aims to synthesize best practice examples from mobile and desktop OS, web applications, and common sense to create a flexible set of UI widgets. Because of the international community and audience for jQuery UI, the project is developed to work in a variety of languages and cultures.

10.4 Findings

In this section we will discuss and describe the methods used to test the hypotheses.

> **H1**: The Linux Kernel community adopts a protective style of governance for its code change process.

We checked to see whether the answers are positive to the questions V27-V30. We show in the diagrams below 10.1 the posterior for answers to questions V27-V30. The plot shows the inferred probability of what answer can be received to each question within the Linux community. The orange line marks the threshold 3, separating positive and negative answers. The horizontal line marks the 95% of the probability mass (HPD).

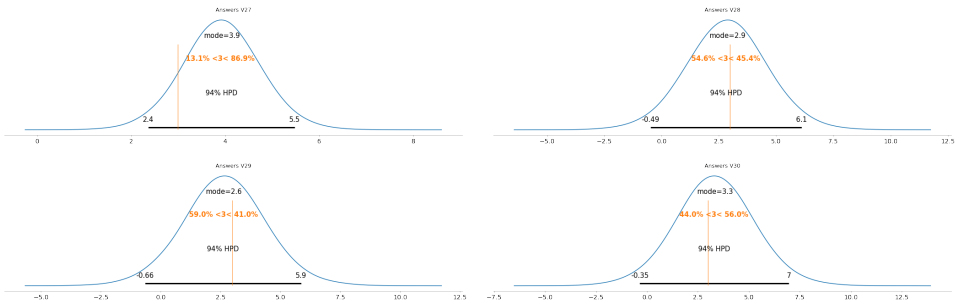

Figure 10.1: The Distribution of the Answers V27-V30 in Linux Kernel Community

We now check the posterior distributions for each question, for the probability of either of the answers being below 3.0, in the lower, so positive, part of the scale. We reject H1 as for none of the questions, the 95% of the probability mass falls into the positive part of the scale. This output suggests that the Linux Kernel community is not protective. The HPD intervals show that the answers are heavily spread over all possibilities, i.e., all values from 1 to 5 are within the HPD interval, and the mode is around the center of the scale. This observation indicates that the community does not show a central tendency toward having a protective style toward pull requests.

Although in **Paper E**, I concluded that Linux is a protective community, I was cautious in reporting this result. I stated that this protective style is visible in some pockets of the Linux communities, while other are less protective. Some of my interviewees conveyed to me that the community is trying to change its attitude toward contributing in general, which could explain the mixed results. The broad part of the HPD above seem to confirm that.

> **H2**: The FOSSASIA community adopts an equitable style of governance for its pull request process.

To study this hypothesis, we estimated the distribution of the answers to questions V31 and V32 by members of the FOSSASIA community. We show the posterior for the expected answer in Figures 10.2 and 10.3 the different distributions for the data. The vertical orange line separates the positive and negative answers.

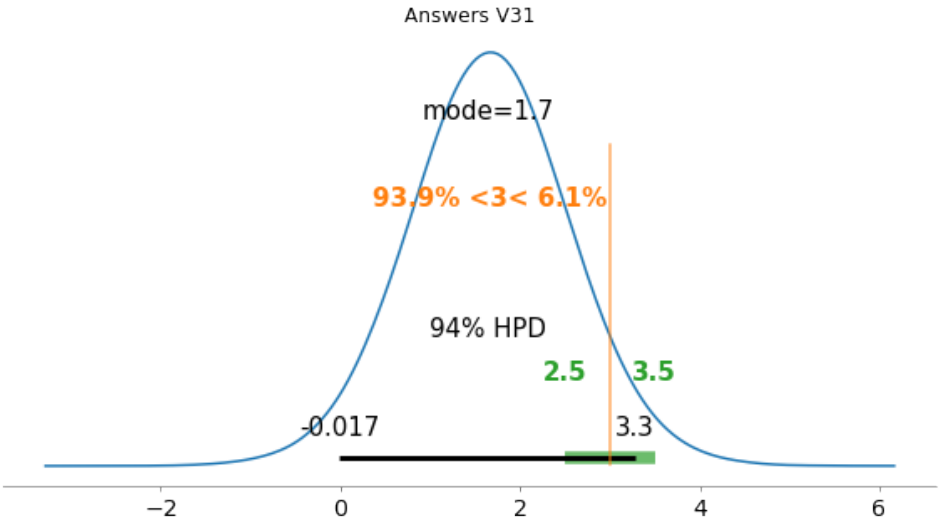

Figure 10.2: The Distribution of the Answers V31 in FOSSASIA Community

We accept H2 as more than 94% of the probability mass falls into the

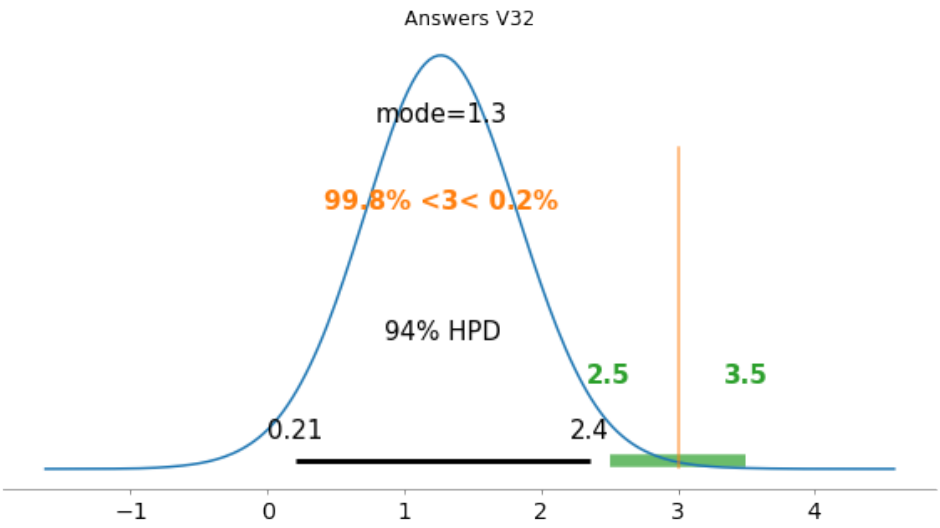

Figure 10.3: The Distribution of the Answers V32 in FOSSASIA Community

positive part of the scale for question V32, even allowing a small margin of error. This margin of error (ROPE) is marked green in the figure.

The distribution for V32 has a mode of 1.3, meaning that dominating answers are strongly positive. The answers to question V31 are closer to neutral. The HDP interval goes from 0 to 3.3, with 6.2% of the distribution is in the range of answers from neutral to disagree, so it is marginally large to accept the answer as positive. This might suggest that there are some participants who responded disagree to this question. These findings are in line with our qualitative component finding, that FOSSASIA is an equitable community. Our interview data shows that despite the majority of our interviewees leaning toward being equitable, there are a minority of maintainers who tend to favor the protective style of governance traits which values the trust and reliability of the contributor.

H3: The Odoo community adopts an equitable style of governance for its pull request process.

We ask whether the Odoo community is equitable by calculating the

probability distributions of answers to V31 and V32. HDPs for both questions are centered around 3 with modes 2.3 and 2.1 respectively. This does not allow to concede that H3 holds. Even though, the probability of the disjunction between the two questions is more concentrated toward positive answers, we cannot classify Odoo as equitable. This does seem to indicate that the Odoo community shows equitable tendencies.

> **H4**: The Coala community adopts a lenient style of governance for its pull request process.

To study this hypothesis, we estimate the posterior distribution of the answers to question V33 by participants of the Coala community. We show the posterior in the Figure 10.4. The HDP of the posterior distribution extends from 0.4 to 3.1, that is most answers go from strongly agree to neutral. Only 3.9% of the answers greater than 3, that is only 3.9% of the answer are on the side of disagree. Therefore, we can conclude that H4 holds. This supports our qualitative study findings that the Coala community exemplifies the lenient style of pull request governance. They strongly believe in being lenient, but in the same time they do not compromise quality, according to our findings.

> **H5**: Each of the 15 FOSS communities adopts a specific governance style, either protective, equitable or lenient, for its pull request process.

To test this hypothesis, we gathered all the answers to all of the questions by all of the communities. We estimated the distribution of the answers to questions V27 to V33 by participants of all communities. As mentioned earlier, we assumed that the data forms a normal distribution, and we set uniform priors on the parameters of the normal distributions for the data, from 1 to 5 for the mean, and 0 to 4 for the standard deviation.

For each community, we plot a bar chart shown in Figure 10.5. The plots draw a green horizontal line showing the threshold of 0.95, shown as the

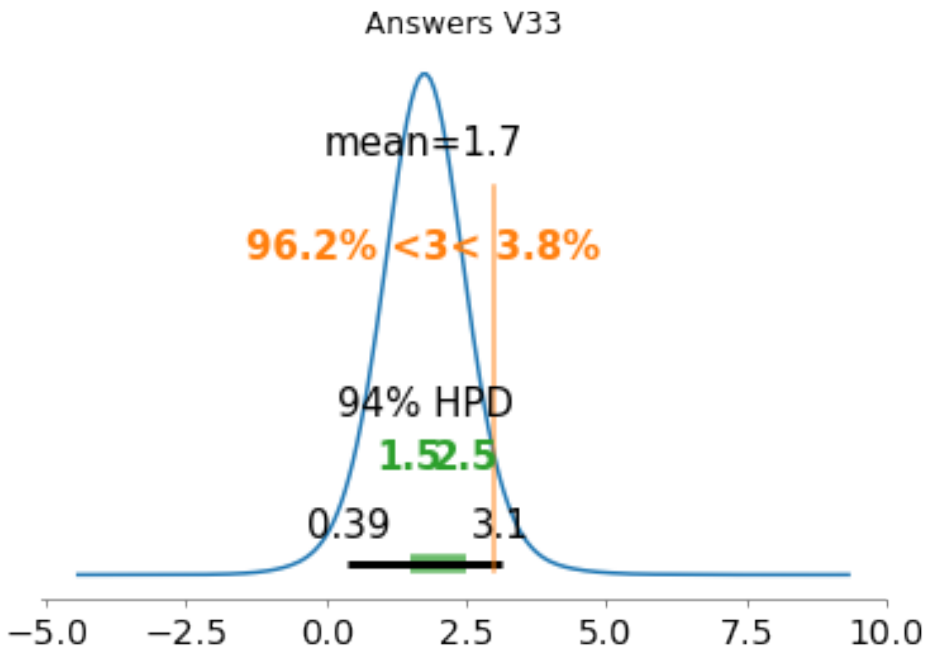

Figure 10.4: The Distribution of the Answers V33 in the Coala Community

condition of the community to be classified as any of the categories.

In order for the hypothesis to be true, for each of the plots above, there should be exactly one bar above the threshold. It is easy to see that this is not the case, as there are some communities that are neither protective nor equitable nor lenient. Also, the Coala community is both lenient and equitable. Therefore, we conclude that H5 is false.

We can conclude that while some communities have a dominant style of governance, other communities are closer to one governance style with tendencies to prefer a second governance style. This behavior is statistically observed in some communities like the Linux Kernel. We can conclude that the Linux Kernel community is slightly more protective than equitable. This finding is supported by the qualitative study as well. Participants in the interviewees conveyed that the community is going through a period of reconsidering its software changes management and evaluation style.

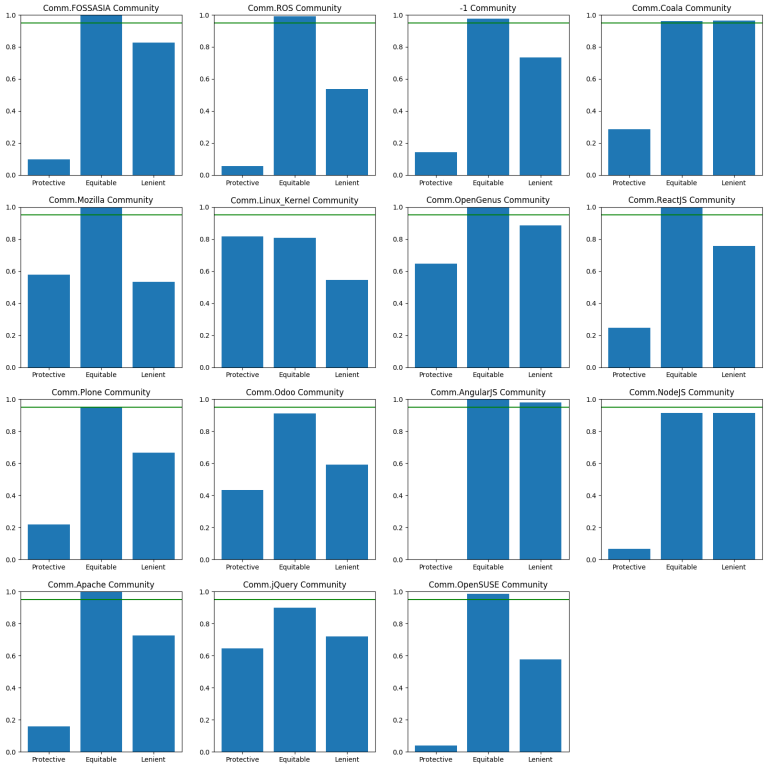

Figure 10.5: Bar chart which showing the probability of a community being protective, equitable and lenient.

To demonstrate that some communities diverge significantly in their governance styles, we propose the following hypothesis:

> **H6**: The Coala Community is more lenient than the Linux Kernel Community.

To study this hypothesis, we estimated the distributions of the answers to questions V33 by the Coala and Linux Kernel communities. We computed the difference between the mean and the standard deviation of the two estimated distributions. Moreover, we computed the effect size between

the two differences. The effect size is a standard method to compare two estimated normal distributions.

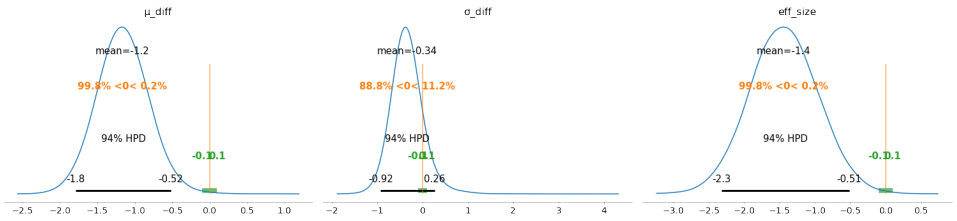

Figure 10.6: Hypothesis 6 Test

We show (Figure 10.6) the distributions on the difference of means, standard deviation, and effect size. These plots provide useful information in determining whether the distributions on the answers are significantly different. The differences of means μ diff indicates that there is a difference of 1.2 indicating a more lenient style, based on the answers from the Coala community. Furthermore, the posterior distribution of the effect size suggests a nonzero difference well outside of a (-0.1,0.1) ROPE. Consequently, we can conclude that H6 holds.

As discussed in H1, the Linux community has protective tendencies. This community values trust and contributor reliability more than other communities do. While the Linux community prefers contributions from trusted and reliable contributors, the Coala community believes that every contribution indicates an enthusiasm from a contributor that should be invested in. When the quality of the contributions does not meet the community standards, the contributor is mentored to elevate her code quality to meet community requirements. The clear difference in the survey data reassures me that the qualitative model has identified relevant phenomenon observed at the large scale as well.

10.5 Discussion

In summary, Hypothesis 1 stating that the Linux Kernel exhibits a protective style was determined to be false as the probability was not 95% or higher. Hypothesis 2 that states that FOSSASIA exhibits an equitable style of viewing pull requests was determined to be true based on the data, as the results were in the region of practical equivalence. Hypothesis 3 stating that the Odoo community exhibits an equitable style was determined to be false as well, as the results were not significant. Hypothesis 4 states that the Coala community exhibits a lenient style, and this hypothesis was determined to be true. Hypothesis 5 states that all FOSS communities adopt a specific style toward pull requests, and the results show that this hypothesis was not supported by the data and is false. Some communities show several styles, and other communities do not indicate a specific style. Hypothesis 6 states that the Coala community is more lenient than the Linux community, and the data analysis determined the hypothesis to be true.

These findings show that FOSS communities are not equal. Each community has different culture, history, leadership and values. These variables likely influence the decision-making mechanisms in the PR evaluation process. The picture emerging from these findings is that most communities have a dominant PR governance style, which exemplify their believes, norms and culture. Communities with no apparent governance style (e.g. Linux Kernel) are likely experiencing a transition period to an emerging governance style.

10.5.1 Dominant Style

The dominant style is the most prevailing style in the community. This is the formal governance style adopted by the community. This style is diffused in the community by its leaders (i.e. usually maintainers and senior contributors). This participant explained, *"we obviously pitch our rules and ways and people learn them ... We believe in being unbiased and instead focus*

on the technical aspects of the contribution" (Participant 5, FOSSASIA, Paper E). However, it is not black and white.

The allocation of a governance style to a community is not always straitforward. As noted in the discussion of H7, some communities have tendencies to prefer a second governance style. This was observed, for example in FOSSASIA community, statistically (Figure 10.5) and qualitatively. A maintainer explained, *"we use to mentor people to improve their code. But we do less of that now. We prefer to be more strict and fair. We do not have enough experienced reviewers ... Some maintainers still prefer to mentor the contributor rather than just rejecting. It [mentoring] is time consuming and we afford it"* (Participant 2, FOSSASIA, Paper E). FOSS communities are reactive to their circumstances and other variables in their environment, they adjust their strategy accordingly. The Coala community, for example, had different attitude and they align their PR governance accordingly. This maintainer explained, *"we prefer to invest in mentoring people to deliver high-quality contributions. Because we believe it [mentoring] helps us recruiting ongoing contributors and in the same time maintaining quality"* (Participant 30, Coala, Paper E). For this community, being lenient is a conscious strategy to sustain participation.

However, being lenient does not imply "drop (one's) guard." A junior contributor is mentored to elevate her contribution to the required quality by a maintainer or a senior contributor. Then, proposed to the community for review and technical inspection. At this stage, the community apply its quality and inspection criteria for evaluation. Hence, H7 plot shows that lenient communities are also equitable. This reflect a dual strategy for PR governance that simultaneously does both, invest in the contributor enthusiasm of submitting a PR and assuring quality by focusing purely on technical grounds, as illustrated in the graphs of Figure 10.5 (see for example Coala, NodeJS and AngularJS).

10.5.2 Emerging Style

An emerging style of PR governance is a style that is becoming prominent in the community, while the old style is still exercised by some members. While the old style has its reasons to exist, the new style is emerging because of a need for change. This was observed statistically in the Linux Kernel community data and explained qualitatively. This contributor explains, *"getting a patch accepted in the Linux community can be difficult for newcomers and unfamiliar names in the community. They have preference for trustworthy contributors. This attitude is cascaded from the top ... However, this is changing. I know a lot of sub-system maintainers who want a community with less fences"* (Participant 25, Linux Kernel, Paper E).

Achieving and maintaining quality is important to these communities. PR governance is a measure to assist in preventing poor quality contributions. Putting structures, direction and control for PRs evaluation to avoid poor code is necessary to enable communities to operate more efficiently, to mitigate risks and safeguard the code base.

Figure 10.7 illustrates this behavior. While the protective and the lenient styles first focus is the person, the equitable style does not give a significant weighing to the person. It prefers to evaluate the PR irrespective of the person. This belief has a rationale: (1) The community believes that fair and impartial to do so and (2) the community has a consistent influx of contributors and contributions. It does not have a need to attract additional contributors. Fairness is exercised by focusing on the technical features of the PR and making decisions that are technically grounded.

The protective style looks for some qualities in the submitter of the PR prior to the evaluation taking place. Hence, it is a two stages process. During the early stage, the gatekeeper asses the trustworthiness of the submitter. This is usually established by the reputation of the person or her relationship with the maintainer. Once this barrier is overcame, then the second stage is

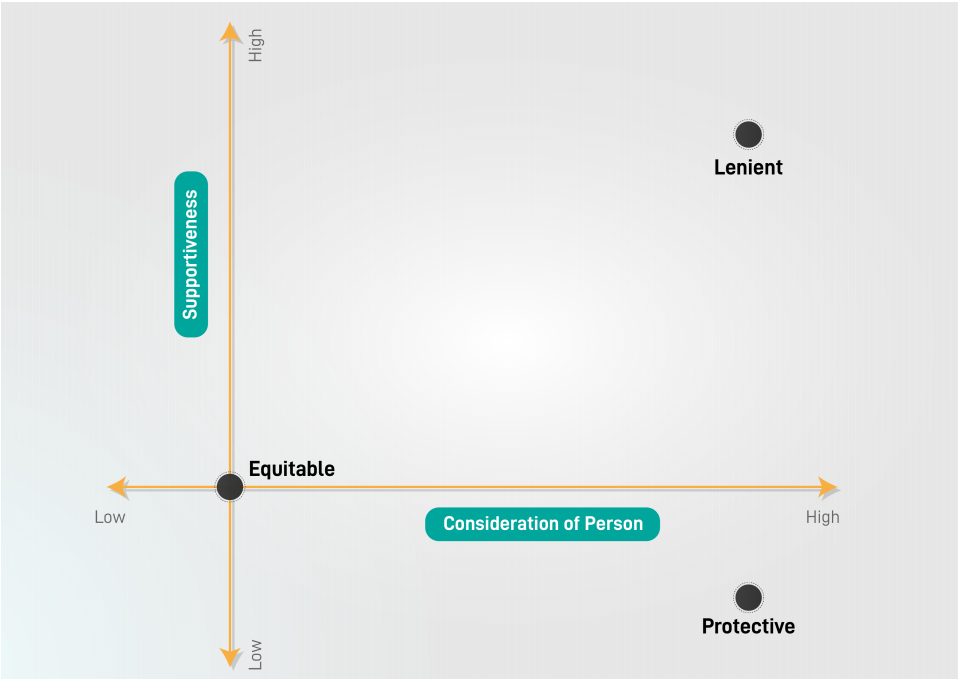

Figure 10.7: FOSS Communities Pull Request Governance Model

to assess the PR technically. This stage is common to all governance styles. The assessment take place prior to the merger. A set of software engineering principles (**Paper E**) are applied to assess the suitability and the quality of the proposed software change.

The lenient style has different philosophy. When a PR is submitted by a newcomer (this always known when a PR labeled for newcomer is submitted and/or the submitter is unknown to the community), then a mentor is assigned to the contributor. This gesture is to avoid rejection. Instead, the community believes that they have to build on the enthusiasm that comes with the submission, hoping that that this investment may convert the contributor to an ongoing contributor.

10.6 Conclusion

These findings show that the studied FOSS communities align their PR governance styles according to their needs and in some instances it evolves from one style to another style. But in all instances, quality is not compromise. What seem to shift is the willingness of these communities to be supportive of the person contributing. The supportiveness of the person varies amongst these styles and it seems to be a reaction to a strategic need. For example, in the case of the Coala community, this strategic need is to sustain participation. On the other hand, the Linux Kernel community is less supportive because it has a strong influx of contributors and it prefers to filter based on trust and relationships building under the pretext that the software is of a level of complexity that necessitate reliable contributors to support the code.

This disciplinary aspect of FOSS communities contribute to achieving quality by mitigating the risks posed by the contributor and the contribution. While the protective style mitigate the risk posed by the person by assessing her trustworthiness, the lenient style prefers to mitigate the risk and mentor the contributor to deliver code within the community standards. The second level of risk mitigation focused on the PR technical quality.

11

Discussion

The rapid development of FOSS as an alternative to traditionally developed software systems leads to a need to examine all of the aspects of the FOSS products and communities. This dissertation is an examination of the social, organizational and disciplinary aspects of quality in FOSS communities. I looked at how these constructs shape and influence delivering quality products in FOSS. I observed that the participants in my studies learn how to deliver quality code and internalize quality. They make part of their attitudes and behaviour; part of one's nature by learning or unconscious assimilation. Quality control relies on developer commitment to the process, which may come from compliance, identification, or internalization [21].

I identified three enablers and two desired features. Enablers are qualities or capabilities that contribute at making quality possible. Desired features are intended capabilities, when achieved they created a desired effect which is maintaining quality. The enablers are (1) personal motivation for quality, (2) governance for quality and (3) the ability to improve. The desired features are (1) active commercial participation and (2) retention to sustain quality. Figure 11.1 illustrates this model. It shows that the combination of the strength of each enabler and desired feature contribute to achieving quality in FOSS communities.

Figure 11.1: Achieving Quality in FOSS Communities

11.1 Personal Motivation for Quality

Quality is a result of systems, techniques, and people. People are the hardest element to control and develop effectively [64]. One of the best ways for quality improvements to occur is through making people more efficient [59]. Individual competence is a critical factor in project success. If the people on a project are competent, they can use almost any process to accomplish their task. Likewise, if the people are inadequate, no process will work [17].

Motivation, that is, initiation, direction, intensity, and persistence, in software engineering is a key success factor for software projects, influencing both quality and productivity [26,102]. Motivation is the software engineering factor reported to have the single largest impact on developers' productivity [116]. When a person is well motivated, he/she can solve problems and achieve work objectives. However, nearly 80% of quality problems are caused by the way people are organized and managed [57].

FOSS contributors are highly motivated to deliver high quality code. This

motivation for quality is usually inherent from the participation motives. Developers are the creators of quality. FOSS contributors are positively motivated to achieve as much perfection in their job as possible. They take pride in their work and use it to build up reputation and community recognition. In addition, they have sense of achievement, enjoyment and responsibility toward their tasks. Motivation for quality in FOSS is also shaped by individual attitudes and perceptions such as the extent of understanding of the purpose and value of quality.

Motivation implies a willingness to make something better. Emotions are states of mind raised by external stimuli, and moods are emotional states where an individual feels good or bad. These terms are often used interchangeably [43]. Happiness is created by high cognitive performance, high motivation, positive atmosphere, higher self-accomplishment, high work engagement, higher creativity, and higher self-confidence, as well as feeling valued or being proud of performance [42].

Affective states are characterized by their valence, arousal, and dominance. Valence, or pleasure, is attractiveness or adverseness of an event, object, or situation; arousal is the sensation of being mentally awake and reactive to stimuli, and dominance is the sensation where the skills are higher than the challenge for a task. Affective states affect work, and describe "flow" as fully focused motivation, energized focus, full involvement, and success in the process of the activity. Happier employees are more productive. There is a strong positive correlation between a positive affective state and task performance and a negative correlation between negative affective states and task performance [43].

One avenue to improve software developers' productivity and software quality is to focus on people. The role of affective states of emotions, moods, and feelings have an impact on cognitive activities [43]. Developers may underperform if they do not feel safe and happy, and positive emotions like happiness make people more creative. Fear can refrain developers from changing their code [76].

The primary characteristics of software developers is that they are growth oriented, that is they like challenges and learning new skills. Another quality is that they are introverted with a low need for social interaction, and another is that they are autonomous. In addition, the literature identified change and challenge as motivators, followed by problem solving opportunities, teamwork opportunities, and a chance to experiment in software development. The task should have clear goals, be interesting to the individual, be clearly defined, and linked to other activities. Also important to software developers is employee participation, good management, career path opportunities, a variety of work, a sense of belonging, and rewards and incentives. Contextual factors such as the individual's personality and the environment also affect the characteristics of engineers and motivational factors [102].

FOSS contributors seem to be happy and motivated to achieve quality in their deliverables. The internalization of quality (i.e. quality become part of the individual believes) combined with motivation for quality (e.g. the quality of one's code is equated to her/his reputation) push the individual to excel in the tasks of software development. Achieving excellence is the result of high performance throughout the entire community. This collective pursuit for excellence produce high quality products.

> ***Section Summary.*** *Quality needs motivation and motivated engineers cherish quality. In chapter 5 (**Paper B**), I established a correlation between FOSS contributors' motivation and quality. FOSS contributors perform code review (a quality task) with high motivation. This motivation contribute at excelling at the task of code review. This claim (motivation positively influence quality) has been supported by software engineering motivation and quality improvement literature.*

Table 11.2 shows the problems addressed by the findings discussed so far in this chapter. Recall **Problem I** highlights the needs to identify and discuss social, organisational and disciplinary aspects of quality. This finding (i.e. FOSS developers motivation for quality is positively correlated to software quality) tackles partially the problem. This finding also addresses **Problem**

II, **Problem III** and **Problem V**. This finding supports empirically the anecdotal evidence that social factors contribute to achieving software quality in FOSS. I demonstrated that FOSS contributors invest their motivation in software development task to achieve quality. Motivation for quality is a social trait that FOSS software adopters should look for in communities as a positive sign for products quality.

11.2 Active Commercial Participation

The sustainability of the open source model has often been questioned because of free-riding, monetization by a party that did not contribute to the open source project, yet does not share its proceeds with the project contributors [80]. Open source models are based on people contributing their time and knowledge in a collaborative effort to create publicly available information goods. Recruiting and motivating contributors is important to open-source communities, and these efforts can mitigate against free-riding.

Nov and Kuk study whether perceived justice of external appropriation will be negatively related to effort withdrawal intentions of FOSS contributors. They find that effort withdrawal resulting from the prospect of free riders is negatively associated with both perceived justice and intrinsic motivation. Perceived justice moderates the effect of intrinsic motivations on free riding. Justice or fairness personality traits moderate the tendency for withdrawal [80]. These findings are strong support to my claim that passive participation is damaging to FOSS.

Most contemporary FOSS communities are hybrids, with commercial and noncommercial interests, motivations, and backgrounds. Company participation and the work ethic it implies brings both dangers and opportunities for long-term sustainability, which includes both vitality and the ability to adapt to changes. Researchers have found that 71% of the most active projects have five or fewer developers, and 51% of them have only one project admin-

istrator [49]. Therefore, these small communities depend on the leadership of the administrator.

Traditional economic theory predicts that the free-rider problem causes inefficient provision of public goods and calls for a central intervention to remedy the free-riding [62]. The equilibrium is determined based on whether the peer's marginal benefit to sharing justifies the cost of sharing. If the cost of sharing is small, there is an equilibrium where everyone shares; but if the costs are high, such that no one shares [62].

Some researchers investigate the motivation for and ability of knowledge exchange, and test the relationship between social capital and knowledge acquisition and contribution. In each period of time, each user independently decides whether or not to share content, and each user demands one unit of content randomly from the other users [62]. Withholding effort is the likelihood that a participant will give less than full effort to a job-related task. It causes shirking, job neglect, social loafing, and free riding. Shirking focuses on the lack of full effort, job neglect focuses on partial withdrawal from job duties, and both focus on an individual working alone. Social loafing is holding back effort, and these occur in group contexts. In the absence of coercion or inducements, individuals tend to withhold knowledge. [65]

Passive participation impacts FOSS ecosystems. My ROS participants conveyed to me their concerns of the sustainability of the community in the increasing passive commercial participation. One participant labeled this behavior and passive participants as "leeches," in reference to the parasitic and predatory worms. These species are predatory, they prey on other creatures in their ecosystem. This comparison is strong and shows the resentment toward this behavior from individual contributors.

> ***Section Summary.*** *In chapter 6 (**Paper C**), I concluded that passive participation negatively affects the sustainability of the ROS community. The sustaining of contributions in FOSS is a concern in an environment with increasing commercial participation. The exploitation of FOSS resources*

by passive participants has negative consequences for quality. Withholding knowledge and efficient contributions lead to de-prioritizing of quality relevant tasks and initiatives.

Table 11.2 shows the identified problems and this dissertation's findings. Each tick in the table cell is an indication that a particular problem is addressed partially or entirely by the relevant findings.

11.3 Ability to Improve

The ability to change and evolve is a key component of software sustainability [82]. Continuous improvement is dependent on the ability to see things in new ways, gain new understanding, and produce new patterns of behavior, on a way that engages the organization as a whole [39]. Achieving sustained quality improvement requires commitment from the entire organization, particularly from top-level management. However, in the case of a community, commitment means engaging the community to bond around the initiative of quality improvement. I used the ROS community to test the ability of FOSS to embrace and implement change. I learned from this engagement with the ROS community that when the motivation for quality exist in addition to the availability of resources to implement the interventions, then change can materialize successfully.

Gasston and Halloran argue that the benefits of implementing quality improvement initiatives is beyond the immediate success of the endeavor itself [39]. They explain that the benefits from implementation of software process assessment and improvement programs will not be successful until organizations move toward becoming learning organizations, organizations which improve knowledge and understanding of themselves and their environment, organizations skilled at creating, acquiring, and transferring knowledge and modifying their behavior (i.e. becoming a learning organization) [39]. Learning organizations depend on systematic problem solving, experimen-

tation with new approaches, learning from their own experience and past history, learning from the experiences of others, and transferring knowledge quickly and efficiency throughout the organization. The method proposed in this dissertation, PAR4FOSS creates such knowledge and learning opportunities. All artifacts, discussions and decisions were published in the community forum to become a recorded knowledge in the community infrastructure.

However, successful interventions and the creation of knowledge are only partly responsible for creating quality improvements. Sustaining these improvements and creating ongoing continuous improvements process is another story. Sustainability, holding the gains of an improvement project, is an important part of a quality program. Quality improvement projects are difficult to sustain after the initial enthusiasm is gone, as quality programs require a significant investment in time and effort [104].

We envisaged the need for a sustainable improvement process. Hence the last cycle of interventions plans for implementing a continuous improvement process. The aim of this final intervention is to create a self-manged process, where the community run the process without the leadership of the researcher.

> ***Section Summary.*** *I designed a method (chapter 7,* ***Paper D****) to evaluate the effect of software quality process improvement in the ROS community using participatory action research (PAR). Before instituting quality measures, I was challenged by the question how to implement change in a FOSS community, determining what is important for a community, and determining how a community can maintain quality. To ensure a fit, I had to align the adaptation of PAR with the community values, beliefs, and norms. I was able to conclude that improving quality practices and introducing change to a community is feasible and the ROS community embraced the change positively. However, the motivation for quality and resources availability are fundamental for such initiative to succeed.*

Table 11.2 shows the progress of the mapping of the defined problems and the findings of this dissertation. Ability to improve addresses **Problem IV**

and **Problem V**.

11.4 Governance for Quality

Governance is a matter of principles [88]. Governance is achieving the direction, control, and coordination of autonomous individuals and organizations on behalf of a FOSS community [69]. The aim of FOSS governance is solving collective action problems, solving coordination problems in software development, and creating a better climate for contributors [69].

Software development and version control services using Git, like Github are used to track, trace, and archive the complete development process in FOSS software communities. Governance of FOSS communities differs with different communities, and version control software is the intersection of governance and coordination. Governance provides the authority in the community, including decision rights, and the coordination between developers on the project. Coordination, which is the common sets of rules, instructions, and activities that operationalize a structure, is embedded in the software tools used in version control and in the social structure of the community as reflected in communication patterns. Volunteers in the community rely on different motivations, and the governance of the community provides support for the various reasons for contributing. Governance is a configuration of multiple authoritative structures embedded in a coordination process that guide activities, tasks, motivations, and effort toward a goal [101]. Governance and coordination are a duality, where the two concepts are distinct and yet strengthen, corroborate, and refine each other. Governance types emerge as coordination works to definitely operationalize rules, guidelines, and activities that define the community. Informal guidelines also work in a community, and these are accessible through the ideologies of the community.

In FOSS communities, expertise, role status, and authority are acquired over time through participation. It is a socialization process nurturing an

identity and learning the practices of the specific platform. Situated learning means that newcomers find mentors to support them in developing the skills and understanding of policies and norms through participation. Each community has its own tools, standards, and protocols, and recognition by other members is through expertise, as well as political influences. Reviewing contributions is not totally objective, but are also politically and socially influenced [88].

Shaikh and Henfridsson identified three different structures; with one of these being centralized authority, reflecting a shared understanding of the central core of key developers as knowing best how to manage and coordinate work. In centralized authority-based communities, peripheral members support the core developers. Another type is libertarian authority, where individual level freedom is imperative, and all members of the community can act autonomously and voice their opinions. A third type is collective authority, where shared understanding is that the needs of the many outweigh the individual right to speak [101]. This behavior is cascaded to the lowest levels of the hierarchy in the community. This cascading effect influences contributors and evaluators behavior during the process of PR evaluations (i.e PR governance).

Unique features of FOSS projects make it possible to have effective behavior, clan, and self-control mechanisms. These mechanisms were higher in coordinated configurations, implying that these mechanisms are complementary. With no contracts or financial incentives, these mechanisms are still effective [30]. PR governance in FOSS communities has three purposes: to encourage collaboration, to coordinate the software development process, and to create an effective developer climate that attracts, motivates developers and monitor the quality of the proposed software changes.

> ***Section Summary.*** *PR governance is the action of controlling the code change process by using rules obtained from the community set of values and norms to influence, direct and control the actions of managing code changes. One of the functions of this governance is to assure quality.*

> *FOSS communities do govern the PR process using different styles (i.e. Protective, Equitable and Lenient) as shown in chapter 8 (**Paper E**). These styles reflect the cultures of the community, its history, leadership type, and prevailing thinking in the community.*

As shown in Table 11.2, this finding addresses **Problem I, Problem II, Problem III** and **Problem V**.

11.5 Retention to Sustain Quality

New developers typically begin by submitting patches or pull requests, which are sets of modifications to a project's code. The submission of a patch is a process that affects the quality and growth of the FOSS system. It is important because it is a primary quality assurance mechanism for FOSS systems, it enables knowledge transfer and learning, and it is an opportunity for recruiting potential developers for a project. The process begins with patch creation, then moves to publication, discovery, review, and release. A project's ability to attract and retain developer resources and active user resources will have a positive effect on its future sustainability [15]. The PR evaluation process can have a significant impact on contributor motivation. The contributor's experience during the evaluation process has further reaching than just whether the contribution is accepted or rejected. Most contributors perceive the process as unpredictable [55]. To reduce this unpredictability and improve the contributor's experience, I proposed seven good practices for the PR evaluation process: engagement, communication, appropriateness, simplicity, compliance, support, and honest decision support the evaluation process positively. Creating positive experience for the contributor hopefully would enhance retention.

One characteristic of FOSS projects is the large number of volunteers involved in the project development. The work is not assigned, and anyone can choose any task that they wish. The teams self-organize their work.

The patch or pull request submission process is one of the most important activities in FOSS development for many reasons. One of these reasons is that it is a mean for patch contributors to demonstrate their technical skills and commitment. The patch contributors can gain more central roles in projects, so it propels the sustainability of the FOSS project. However, there are barriers to FOSS contributions as well. The patch review varies between FOSS projects, so this can cause confusion and require time to learn the system for a particular community. In addition, patch review is slow and requires a lot of time, and patches are often lost or ignored. The unreviewed patch number runs from 27% to 54% in some projects [100]. Then the majority of patches are rejected, with more than half of submitted patches falling into this category. There are more patch contributors than reviewers as well [100].

The pull request is the central work product in the modification of the FOSS software [100]. PR evaluation consists of verifying the submitted code, which checks for quality, security, maintainability, integration, testing, and licensing; refining the patch, which is using feedback to modify the patch; and resolving the patch, which is the final outcome, either accepting or rejecting the patch. Although rejected patches can be resubmitted, some are rejected numerous times. Patch application is when a committer uses available tools to apply the patch to the code repository.

Better-managed projects increase the chances of sustainability. Developers are willing to follow project leaders as long as the project leader listens to developers' views [83]. A critical task of sustaining the platform is to ensure an adequate number of participants, which are the most valuable resource. People join these communities for intrinsic and extrinsic reasons. Fostering an environment that is open to new contributors is important [88]. Openness of social coding creates transparency, but both technical and social factors affect the chance of acceptance [120].

literature on volunteers management looked at retention extensively and FOSS research should leverage some of the findings, as there are strong

similarities. There is a consensus in this literature that volunteers who could leave their position as a volunteer were more likely to do so [22, 73, 79, 92, 117]. When altruistic reasons for volunteering are not satisfied, volunteers are less likely to remain with the organization [73]. Volunteer retention occurs when motives for joining are satisfied, and when alignment with goals and values exists [117]. Values such as inclusivity, and fairness enhances the retention of volunteers. Training and support were identified as the most important reasons for volunteers retention. Other reasons given were organizational support, social networking, positive job characteristics such as nonrepetitive tasks, jobs with gratifying tasks and clear objectives, benefiting others, and training volunteers [79].

In FOSS, the probability of a newcomer becoming a long term contributor was found to be associated with the person's extent of involvement and interactions with her environment [119]. The extent to which an individual's values are consistent with the community's impacts how long they stay on the project. Identity and bonds in the community also affect their commitment to the project [119]. Pull request evaluation is one of the major avenues of exposure to the community believes and behavioral systems. Through this avenue communities should display and promote an attitude based on support and fairness. This should enhance the contributor's experience and consequently a better resources retention. I proposed a framework that advocate a set of good practices for every stakeholder in the PR evaluation process. Table 11.1 lists the good practices and the corresponding stakeholder(s) responsible to adhere to the practice.

- Engagement: As indicated previously, the literature suggests that unreviewed and neglected PRs in FOSS communities is considerably high. This can be demotivating for a contributor to see her or his work being unappreciated. My conclusion after analyzing 58 pull requests is that FOSS contributors relate fairness to community engagement around the PR. Engagement occur when a timely and an adequate

Good Practices	Contributor	Maintainer	Community
Engagement			✓
Communication	✓	✓	✓
Appropriateness	✓		
Simplicity	✓		
Compliance	✓		
Support			✓
Decision		✓	

Table 11.1: PR evaluation practices and involved stakeholders

number of reviewers attend to the evaluation of the PR. The enthusiasm and the effort invested in writing and submitting a PR should not be met by a failure to care.

- Communication: I observed that PRs considered being assessed fairly exhibit professional and camaraderie style of communication. However, those deemed unfairly evaluated, some of them display hostility and harsh language. This lead me to conclude that contributors appreciate a professional and friendly communication during the evaluation of their PRs.

- Appropriateness: Most FOSS communities I studied have a vision for their products and sometimes this vision is well documented. Contributors are required to submit changes that adhere to the overall product vision of the community. When, they fail to do so, it either creates conflict or simply the PR is neglected by the community. Contributors should accustom themselves with their community products' road-maps which (often documented in file VISION.MD) and submit PRs within the scope of the vision document. I have noticed that PRs within the scope of the community road-maps receive adequate engagement and enthusiasm from the community.

- Simplicity: FOSS communities strive for simplicity. It allows them control over the software code and it facilitates speedy reviews of code changes. Complex PR are regarded risky and time consuming to review.

They usually face long diverging discussions without a conclusion. They end up abandoned or rejected. My data shows that PRs adhere to simplicity principle are more likely to avoid this entanglement.

- Compliance: FOSS communities do not like barrages of rules, but when they have rules they like to enforce them and appreciate when contributors adhere to them. This practice mainly calls for contributors to be mindful and stick to the community guidelines for code styling, architectural decisions and codes of conduct.

- Support: This is the assistance that the contributor receives during the PR evaluation process to meet the reviewers requirements for quality. I have observed that PRs receiving good community support from mentoring and advise were considered fairly assessed. Contributors feel that support received from the community is a reward for their effort.

- Decision: Eventually, after the reviewers accept the PR, it is the committer decision to either adhere to the reviewers recommendation or overrule it. This is a critical point in the process. My respondents made it clear that fairness should be exercised in the stage of the process. Failure to do so is a risk to lose a contributor.

Volunteers quit when they feel undervalued [92]. Minimizing attrition through retention practices is necessary. This can be done through providing good experience for the contributor and increasing morale. Contributors' perceptions of the community, fairness, and the degree of recognition increase satisfaction amongst contributors. Community cohesion aids in job satisfaction as well. The framework of social exchange theory states that to keep volunteers, the rewards must exceed or balance out the costs. Volunteers assess the relative rewards and costs of their involvement, and this assessment determines if they will remain volunteering [6, 92]. Conflict and dysfunction in a work group can erode social benefits and cause volunteers to resign.

Problems	Brief Descriptions	Findings & Solutions
Problem I	Need to understand the social, organizational and disciplinary aspects of quality.	Personal Motivation for Quality, Active Commercial Participation, Governance for Quality, and Retention to Sustain Quality
Problem II	Lack of empirical evidence that quality in FOSS is achieved by additional social aspects	Personal Motivation for Quality, and Governance for Quality
Problem III	How do social and organizational aspects of FOSS contribute to achieving quality in FOSS?	Personal Motivation for Quality, Active Commercial Participation, Governance for Quality, and Retention to Sustain Quality
Problem IV	How can communities leaders steer their communities quality to positive direction?	Personal Motivation for Quality, Active Commercial Participation, Ability to Improve,
Problem V	What are the positive signs of quality in FOSS communities?	Active Commercial Participation, Ability to Improve, Governance for Quality, and Retention to Sustain Quality

Table 11.2: The problems and their corresponding solutions

Section Summary. *Quality needs resources. Then retention of resources is important. Dysfunctional turnover is when contributors leave because of working conditions. The best climate for success is a supportive atmosphere that is welcoming with opportunities for growth. Good practices variables such as engagement and communication, facilitating the development of positive social interaction with peers and equitable treatment are an important part of good working conditions and enhance the contributor journey. Volunteers management literature support my claim that enhancing the contributor's experience facilitate better retention.*

Table 11.2 shows the mapping of this dissertation findings and solutions to the problems identified in Chapter 2. This finding satisfies the problems

described in **Problem I**, **Problem II**, **Problem III** and **Problem V**

11.6 Chapter Summary

Software development process is not purely a technical task, but a complex "psycho-socio-technical" activity influenced with organizational, cultural, and social structures [3, 25]. Assuring quality in FOSS development is not purely technical either (**Paper A**). In this chapter I discussed some of the non-technical aspects of achieving quality in FOSS communities. Social, organizational and disciplinary enablers and desired features make quality possible in FOSS. Then, how? The personal motivation for quality (**Paper B**) is a characteristic in FOSS contributors. They are highly motivated to write high-quality code. It is their pride and instrument to build reputation in the community. This personal motivation is invested in the task of code review which leads to high performance and excellence in the review process and consequently higher code quality. The sustainability of the community is essential to achieving quality in FOSS. However, FOSS sustainability is negatively affected by passive commercial participation (**Paper C**). It is a desired feature to have in FOSS is active participation of commercial entities. Ability to change and embrace best practices and tools (**Paper D**) is another desired feature in FOSS to achieve quality. The ROS community has shown that a FOSS community can collectively collaborate and implement quality improvement initiatives. Another enabler of quality is PR governance. The decision-making process of PR merger entails exercising the community beliefs, values and culture in choosing a decision. Quality is the primary driver to control the PR process (**Paper E**). A community with lack of resources to contribute ends up with orphan and abandoned PRs, without enough contributors to review the code. Following the seven good practices may help to create a supportive environment during the evaluation of PRs (**Paper F**).

12

Conclusion and Future Work

As quality is a difficult concept to define in software development, it is a difficult concept to examine. Quality is difficult to define because it is people and criteria dependent. This study is an investigation into the quality in FOSS. FOSS is unique in its implementation of software development processes and in the self-assignment of work. FOSS is a viable option for high quality software, in spite of the geographic distribution of FOSS teams and often unpaid work of the volunteer contributors.

One reason for the high quality of FOSS is the openness of the entire project, including code and documentation, which allows for more feedback that can be used for improvements. In addition, developers' motivation is higher and projects are able to attract talented, dedicated individuals. So far, most of these claims were anecdotal put forward by the pioneers of open source (e.g. Linus Torvalds, Richard Stallman, Eric Raymond, etc.). We need empirical evidence to understand how quality is achieved in FOSS. To support this need, I investigated this question: How do social, organizational and disciplinary factors contribute to maintaining software quality in FOSS Communities?

The underpinning epistemological paradigm of this research is pragmatism. Pragmatism is a deconstructive paradigm that advocates the use of mixed methods in research. Pragmatism includes freedom of inquiry in which researchers can define issues that matter most to them and the community and pursue those issues in meaningful ways. Pragmatism includes rejecting

skepticism, the willingness to accept that we are fallible, and realize that sharp dichotomies do not exist, but most beliefs are on a continuum. Inquiry that is based on pragmatic foundations embraces both ideas and actions.

I concluded that achieving quality is influenced by social, organizational and disciplinary factors. So, how? These factors are able to affect community members attitudes, opinions, behavior and interests, thereby directly impacting the software development activities, including quality related tasks. I identified three quality enablers. Enablers are the behaviours and the believes, the tools and resources that will enable a community to achieve quality.

- **Personal Motivation for Quality (Social Enabler)**: One of the major factors in producing quality software, in FOSS communities, is personal motivation for quality. This personal motivation results in FOSS contributors developing exceptional software and excelling at tasks like code review, which leads to higher quality deliverables. The motivation to create quality software is internalized, that is, it is an attitude. The desire to develop a quality product is internalized, a value that is learned through assimilation.

- **Governance for Quality (Disciplinary Enabler)**: Pull Request Governance is defined as the rules in the social system of a community to evaluate code changes and adopt decisions. The role of governance is achieving quality. There are three styles of governance (i.e. Protective, Equitable and Lenient) used in the pull request process that is part of FOSS communities. These governance styles contribute to a FOSS community achieving quality products and control of the process.

- **Ability to Improve (Organizational Enabler)**: In a study of the ROS community (Paper A), I observed that contributors in ROS have relatively little motivation for quality. To improve quality control efforts, I implemented PAR4FOSS, a model for change derived from participatory action research. This action showed that a FOSS community can improve its quality practices when the resources to assist in the

implementation are available, and the contributors are motivated to participate in quality control efforts.

Achieving quality in FOSS requires the satisfaction or the implementation of some desired features in the community. These are discussed below:

- **Active Commercial Participation (Organizational)**: Participation in FOSS communities by commercial entities is increasing. Some of the time this participation is passive, inbound only. When companies fail to contribute as well, the sustainability of the community is impacted, as well as the quality of deliverables. Active participation of commercial entities is necessary to ensure an equitable use of the community resources and a sustainable growth.
- **Retention of Participants to Sustain Quality (Social)**: Most of the interaction with the community and those wishing to join the community occurs in the pull request process. Therefore, an analysis of the pull request process is the salient point to examine the presence of fair treatment of those submitting pull requests and fair assessment of their contributions. Seven practices are recommended for the pull request process that help to sustain the retention of FOSS communities contributors.

So what? Our view of achieving quality as a purely technical activity needs to change. Achieving quality is an engineering, social, organizational and disciplinary activity simultaneously. Our effort in researching quality in software engineering should be equally distributed across these aspects. We need to further understand how non-engineering aspects contribute to achieving quality in software development teams and FOSS communities.

Another implication is how managers and software development organizations implement quality measures in their processes. Non-technical aspects and technical aspects are inherently cobbled together. Measures like the

engineers motivation for quality, retention of engineers, and continuous improvement should be given equal status in software development processes as the engineering aspects.

We need to educate software engineers that quality is a belief and the quality of their work is a pride and has consequences in their reputation. Education institution should teach quality as a professional value rather than an item in a checklist. In software engineering educational programs quality should be put as a value front and center.

So how can employers make quality as a value stick? Reminding engineers of values does not stop after crafting, laminating and posting posters throughout the office, however, they need to be communicated from the top on a regular basis. Building a workforce that lives and works by the company values begins with hiring based upon values. For each of the company's values, develop a list of questions designed to assess a candidate's character and potential fit and quality should be one of them. Software development organizations should promote quality as an organizational value by rewarding behaviors that demonstrate it. Managers should not hesitate to publicly reward engineers for exhibiting behaviors that are in line with the company's character. I learned from studying FOSS communities, that it not only does make the individual feel good, it also pushes the rest of the team to follow suit.

To conclude, I will summarize the discussion and the rationale of my theses. I proposed five theses (see Chapter 2):

- **T1** A number of human and social aspects create a psychological and social environment that drives the contributors to excel and collaboratively produce high quality code. This was argued and validated in **Paper B**. I also discussed this in Chapter 11. I proved a correlation between FOSS contributors' personal motivation and achieving quality.
- **T2** Active commercial participation in FOSS enhances the sustainability of quality. This was argued and validated in **Paper C**. This has

also been discussed in Chapter 11. I demonstrated that the sustainability of FOSS communities is affected by passive participation which consequently affected quality related task and deliverable.

- **T3** I argue that FOSS communities have the ability collaborate and invest effort in implementing change. To be able to evolve and continuously improve quality practices is a demonstration that FOSS communities can pursue achieving software quality. This was argued and validated in **Paper D**. I demonstrated that the ROS community has the ability to engage in an endeavor to enhance its quality processes.

- **T4** Each community adopts a governance style to oversee the quality of the suggested changes to the code base. Having in place a controlled form of behaviour or way of working contributes to maintaining software quality. This was argued and validated in **Paper E** and Chapter 10. To mitigate the risk of poor code quality, FOSS communities put in place a governance style to assure quality.

- **T5** I argue that the contributor's journey should be enhanced by good PR practices to improve contributors retention. This was argued and validated in **Paper F**. In Chapter 11, I argued that there is a significant link between retention and quality.

12.1 Future Work

While this dissertation reveals interesting patterns, it is clear that further work remains to be done. Below, I discuss some future research directions.

Collaborative effort impact on software quality: Free and open source software (FOSS) changes the way software is developed, perceived, and distributed. FOSS is a collaborative effort by volunteers to create high quality software. Free and open source software (FOSS) changes the way software is developed, perceived, and distributed. FOSS is a collaborative effort by

volunteers to create high quality software. This aspect is still neglected in the software engineering research, i.e. how does the collaborative effort of FOSS contributors contribute to achieving quality in FOSS?

Organizational Culture and Quality: Quality in FOSS communities is a culture and a belief. Quality is internalized by newcomers via different mechanisms (e.g. mentoring, harsh feedback, etc.). Why does not the same occur in close source environments? Organizational culture is directly connected with effectiveness and performance of the organization. I suggest to evaluate the organizational aspects of software-intensive organizations and examine the impact of such culture of achieving software quality.

The Power of the Maintainers and its impact on quality: I observed that maintainers in FOSS enjoy certain power that was usually awarded for their past effort, reputation and technical excellence in the community. I was told by some maintainers that they sometimes override the community decisions on the quality review of code. Power and influence are fundamental human phenomena that are deeply ingrained on the psyche and conscious personality of individuals. The difference between proper and improper use of power is the difference between success and failure, high and low productivity, motivation and disillusionment. I suggest to investigate how do maintainers exercise their power? And do maintainers styles of exercising power impact quality in FOSS?

What is the working style suitable for software engineers? I interviewed contributors who dedicate their free time at night after their daytime jobs to their FOSS communities. They explained that they do it out of passion for their community. In FOSS, there is a long-held belief in meritocracy, or the idea that the best work rises to the top, regardless of who contributes it. Other FOSS traits also appeal to these type of contributors, such as self-assignment of tasks. This may indicate that software engineers have preferences for a community-like style of management. For example, no supervision, self-assignment of tasks, meritocratic system, etc. It would be interesting to investigate the style of management preferred by software

engineers.

Bibliography

[1] Navid Ahmadi, Mehdi Jazayeri, Francesco Lelli, and Sasa Nesic. A survey of social software engineering. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering-Workshops*, pages 1–12. IEEE, 2008.

[2] Robert C Allen. Collective invention. *Journal of Economic Behavior & Organization*, 4(1):1–24, 1983.

[3] Vincenzo Ambriola et al. *Software Process Technology: 8th European Workshop, EWSPT 2001 Witten, Germany, June 19-21, 2001 Proceedings*, volume 8. Springer Science & Business Media, 2001.

[4] Gary Anthes. Open source software no longer optional. *Communications of the ACM*, 59(8):15–17, 2016.

[5] Richard Baskerville and A Trevor Wood-Harper. Diversity in information systems action research methods. *European Journal of information systems*, 7(2):90–107, 1998.

[6] Tony Baxter-Tomkins, Michelle Wallace, et al. Recruitment and retention of volunteers in emergency services. *Australian Journal on Volunteering*, 14:39, 2009.

[7] David Bretthauer. Open source software: A history. *University of Connecticut*, 2001.

[8] Yuanfeng Cai and Dan Zhu. Reputation in an open source software community: Antecedents and impacts. *Decision Support Systems*, 91:103–112, 2016.

[9] Brendan G Cain, James O Coplien, and Neil B Harrison. Social patterns in productive software development organizations. *Annals of Software Engineering*, 2(1):259–286, 1996.

[10] Martin Campbell-Kelly. Historical reflections will the future of software be open source? *Communications of the ACM*, 51(10):21–23, 2008.

[11] Eugenio Capra, Chiara Francalanci, Francesco Merlo, and Cristina Rossi Lamastra. A survey on firms' participation in open source community projects. In *IFIP International Conference on Open Source Systems*, pages 225–236. Springer, 2009.

[12] Kathy Charmaz. Premises, principles, and practices in qualitative research: Revisiting the foundations. *Qualitative health research*, 14(7):976–993, 2004.

[13] Kathy Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.

[14] Brenda Chawner. Community matters most: factors that affect participant satisfaction with free/libre and open source software projects. In *Proceedings of the 2012 iConference*, pages 231–239. ACM, 2012.

[15] InduShobha Chengalur-Smith, Anna Sidorova, and Sherae Daniel. Sustainability of free/libre open source projects: a longitudinal study. *Journal of the Association for Information Systems*, 11(11):657, 2010.

[16] Jan Chong, Robert Plummer, Larry J Leifer, Scott R Klemmer, Ozgur Eris, and George Toye. Pair programming: When and why it works. In *PPIG*, page 5. Citeseer, 2005.

[17] Alistair Cockburn and Jim Highsmith. Agile software development, the people factor. *Computer*, 34(11):131–133, 2001.

[18] Melvin E Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[19] John W Creswell, VL Plano Clark, and AL Garrett. Advanced mixed methods research. *Handbook of mixed methods in social and behavioural research. Thousand Oaks, CA: Sage*, pages 209–240, 2003.

[20] Philip B Crosby. *Quality is free: The art of making quality certain*, volume 94. McGraw-hill New York, 1979.

[21] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):1–35, 2008.

[22] Graham Cuskelly. Volunteer retention in community sport organisations. *European sport management quarterly*, 4(2):59–76, 2004.

[23] Linus Dahlander and Mats G. Magnusson. Relationships between open source software companies and communities: Observations from Nordic firms. *Research policy*, 34(4), 2005.

[24] Tomi Dahlberg and Janne Jarvinen. Challenges to is quality. *Information and Software Technology*, 39(12):809–818, 1997.

[25] Robertas Damaševičius. On the human, organizational, and technical aspects of software development and analysis. In *Information Systems Development*, pages 11–19. Springer, 2009.

[26] Suzana Candido de Barros Sampaio, Emanuella Aleixo Barros, Gibeon Soares de Aquino Junior, Mauro Jose Carlos e Silva, and Silvio Romero de Lemos Meira. A review of productivity factors and strategies on software development. In *2010 fifth international conference on software engineering advances*, pages 196–204. IEEE, 2010.

[27] Cleidson De Souza, Jon Froehlich, and Paul Dourish. Seeking the source: software source code as a social and technical artifact. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 197–206, 2005.

[28] Benoit Demil and Xavier Lecocq. Neither market nor hierarchy nor network: The emergence of bazaar governance. *Organization studies*, 27(10):1447–1466, 2006.

[29] Norman K Denzin and Yvonna S Lincoln. *Strategies of qualitative inquiry*, volume 2. Sage, 2008.

[30] Dany Di Tullio and D Sandy Staples. The governance and control of open source software projects. *Journal of Management Information Systems*, 30(3):49–80, 2013.

[31] Yvonne Dittrich. Quality assurance process and community management in ros.

[32] Neil F Doherty and Malcolm King. The importance of organisational issues in systems development. *Information Technology & People*, 1998.

[33] Evert Eckhardt, Erwin Kaats, Slinger Jansen, and Carina Alves. The merits of a meritocracy in open source software ecosystems. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, pages 1–6, 2014.

[34] M Elliott and Walt Scacchi. Communicating and mitigating conflict in open source software development projects. *Projects & Profits*, pages 25–41, 2002.

[35] Margaret Elliott. Examining the success of computerization movements in the ubiquitous computing era: Free and open source software movements. *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing, Information Today, Inc., to appear*, 2008.

[36] Margaret S Elliott and Walt Scacchi. Free software developers as an occupational community: resolving conflicts and fostering collaboration. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 21–30. ACM, 2003.

[37] Brian Fitzgerald and Tony Kenny. Developing an information systems infrastructure with open source software. *IEEE Software*, 21(1), 2004.

[38] David A Garvin and WD Quality. What does product quality really mean? *Sloan management review*, 25, 1984.

[39] Jennifer Gasston and Pat Halloran. Continuous software process improvement requires organisational learning: An australian case study. *Software Quality Journal*, 8(1):37–51, 1999.

[40] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014.

[41] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 285–296. IEEE, 2016.

[42] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. What happens when software developers are (un) happy. *Journal of Systems and Software*, 140:32–47, 2018.

[43] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. Are happy developers more productive? In *International Conference on Product Focused Software Process Improvement*, pages 50–64. Springer, 2013.

[44] David Hales and Chris R Douce. Modelling software organisations. In *PPIG*, page 13, 2002.

[45] Brian Hanks. Empirical studies of pair programming. In *Position paper for EEAP'03-2nd International Workshop on Empirical Evaluation of Agile Processes*, 2003.

[46] Stephanie Harvey-Jordan and Sarah Long. The process and the pitfalls of semi-structured interviews. *Community Practitioner*, 74(6):219, 2001.

[47] Øyvind Hauge, Claudia Ayala, and Reidar Conradi. Adoption of open source software in software-intensive organizations–a systematic literature review. *Information and Software Technology*, 52(11):1133–1154, 2010.

[48] Frank Hecker. Setting up shop: The business of open-source software. *IEEE software*, 16(1), 1999.

[49] Nina Helander and Maria Antikainen. Essays on oss practices and sustainability.

[50] Pekka Himanen. *The hacker ethic*. Random House, 2010.

[51] Pamela Hinds and Cathleen McGrath. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 343–352. ACM, 2006.

[52] Project Management Institute. *A guide to the project management body of knowledge (PMBOK guide).*, volume 2. Project Management Inst, 2000.

[53] George Issac, Chandrasekharan Rajendran, and RN Anantharaman. Determinants of software quality: customer's perspective. *Total Quality Management & Business Excellence*, 14(9):1053–1070, 2003.

[54] Chris Jensen and Walt Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings of the 29th international conference on Software Engineering*, pages 364–374. IEEE Computer Society, 2007.

[55] Yujuan Jiang, Bram Adams, and Daniel M German. Will my patch make it? and how fast? case study on the linux kernel. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 101–110. IEEE, 2013.

[56] Joseph Juran and A Blanton Godfrey. Quality handbook. *Republished McGraw-Hill*, 173(8), 1999.

[57] Gopal K Kanji. Quality motivation. *Total Quality Management*, 6(4):427–434, 1995.

[58] Mohamad Kassab, Joanna F DeFranco, and Phillip A Laplante. Software testing: The state of the practice. *IEEE Software*, 34(5):46–52, 2017.

[59] Dennis F Kehoe. *The fundamentals of quality management*. Springer Science & Business Media, 2012.

[60] Stephen Kemmis. Participatory action research and the public sphere. *Educational action research*, 14(4):459–476, 2006.

[61] Stefan Koch. Evolution of open source software systems–a large-scale investigation. In *Proceedings of the 1st International Conference on Open Source Systems*, pages 148–153, 2005.

[62] Ramayya Krishnan, Michael D Smith, Zhulei Tang, and Rahul Telang. The impact of free-riding on peer-to-peer networks. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10–pp. IEEE, 2004.

[63] John K Kruschke and Torrin M Liddell. Bayesian data analysis for newcomers. *Psychonomic bulletin & review*, 25(1):155–177, 2018.

[64] Agnieszka Kujawińska, Katarzyna Vogt, and Adam Hamrol. The role of human motivation in quality inspection of production processes. In *Advances in Ergonomics of Manufacturing: Managing the Enterprise of the Future*, pages 569–579. Springer, 2016.

[65] Tung-Ching Lin and Chien-Chih Huang. Withholding effort in knowledge contribution: The role of social exchange and social cognitive on project teams. *Information & Management*, 47(3):188–196, 2010.

[66] Synopsys Mel Llaguno. Open source solution manager.[nd]. 2017 coverity scan report. open source software-the road ahead.

[67] Luis F Luna-Reyes, Jing Zhang, J Ramón Gil-García, and Anthony M Cresswell. Information systems development as emergent socio-technical change: a practice approach. *European Journal of Information Systems*, 14(1):93–105, 2005.

[68] Björn Lundell, Brian Lings, and Edvin Lindqvist. Perceptions and uptake of open source in swedish organisations. In *IFIP International Conference on Open Source Systems*. Springer, 2006.

[69] M Lynne Markus. The governance of free/open source software projects: monolithic, multidimensional, or configurational? *Journal of Management & Governance*, 11(2):151–163, 2007.

[70] Lindsay Marshall and Jim Webber. The misplaced comma: Programmers' tales and traditions. In *PPIG*, page 14, 2002.

[71] Mary Maynard. Methods, practice and epistemology: The debate about feminism and research. *Researching women's lives from a feminist perspective*, 10(26):10–26, 1994.

[72] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. The impact of pair programming on student performance, perception and persistence. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 602–607. IEEE, 2003.

[73] Debra J Mesch, Mary Tschirhart, James L Perry, and Geunjoo Lee. Altruists or egoists? retention in stipended service. *Nonprofit Management and Leadership*, 9(1):3–22, 1998.

[74] Martin Michlmayr. Quality improvement in volunteer free and open source software projects: Exploring the impact of release management. 2007.

[75] Matthew B Miles, A Michael Huberman, and J Saldañ. *Qualitative data analysis: a methods sourcebook Newbury Park*. CA, USA, SAGE Publications [Google Scholar], 2013.

[76] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th working conference on mining software repositories*, pages 262–271, 2014.

[77] Kathleen Musante and Billie R DeWalt. *Participant observation: A guide for fieldworkers*. Rowman Altamira, 2010.

[78] Kathryn E Newcomer, Harry P Hatry, and Joseph S Wholey. Conducting semi-structured interviews. *Handbook of practical program evaluation*, 492, 2015.

[79] Cameron Newton, Karen Becker, and Sarah Bell. Learning and development opportunities as a tool for the retention of volunteers: A motivational perspective. *Human Resource Management Journal*, 24(4):514–530, 2014.

[80] Oded Nov and George Kuk. Open source content contributors' response to free-riding: The effect of personality and context. *Computers in human behavior*, 24(6):2848–2861, 2008.

[81] Alessandro Nuvolari. Open source software development: Some historical perspectives. *First Monday*, 10(10), 2005.

[82] Linus Nyman and Juho Lindman. Code forking, governance, and sustainability in open source software. *Technology Innovation Management Review*, 3(1), 2013.

[83] Linus Nyman, Tommi Mikkonen, Juho Lindman, and Martin Fougère. Perspectives on code forking and sustainability in open source software. In *IFIP International Conference on Open Source Systems*, pages 274–279. Springer, 2012.

[84] Hugo M Ortner. The human factor in quality management. *Accreditation and Quality Assurance*, 5(4):130–141, 2000.

[85] Margit Osterloh and Sandra Rota. Open source software development—just another case of collective invention? *Research Policy*, 36(2):157–171, 2007.

[86] David Lorge Parnas and Mark Lawford. The role of inspection in software quality assurance. *IEEE Transactions on Software engineering*, 29(8):674–676, 2003.

[87] Russell Pavlicek and Robin Foreword By-Miller. *Embracing insanity: Open source software development*. Sams, 2000.

[88] Giacomo Poderi. Sustaining platforms as commons: perspectives on participation, infrastructure, and governance. *CoDesign*, 15(3):243–255, 2019.

[89] Denise F Polit and Cheryl Tatano Beck. Generalization in quantitative and qualitative research: Myths and strategies. *International journal of nursing studies*, 47(11):1451–1458, 2010.

[90] Richard E Potter and Pierre A Balthazard. Understanding human interactions and performance in the virtual team. *JITTA: Journal of Information Technology Theory and Application*, 4(1):1, 2002.

[91] Thiagarajan Ravichandran and Arun Rai. Quality management in systems development: an organizational system perspective. *MIS quarterly*, pages 381–415, 2000.

[92] Simon Rice, Barry Fallon, et al. Retention of volunteers in the emergency services: Exploring interpersonal and group cohesion factors. *Australian Journal of Emergency Management, The*, 26(1):18, 2011.

[93] Colin Robson and Kieran McCartan. *Real world research*. John Wiley & Sons, 2016.

[94] Clive CH Rosen. The influence of intra-team relationships on the systems development process: A theoretical framework of intra-group dynamics. In *PPIG*, page 4. Citeseer, 2005.

[95] Richard M Ryan and Edward L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55(1):68, 2000.

[96] Steve Sawyer and Patricia J. Guinan. Software development: Processes and performance. *IBM systems journal*, 37(4):552–569, 1998.

[97] Walt Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings-Software*, 149(1):24–39, 2002.

[98] Judith Schoonenboom and R Burke Johnson. How to construct a mixed methods research design. *KZfSS Kölner Zeitschrift für Soziologie und Sozialpsychologie*, 69(2):107–131, 2017.

[99] Kristie W Seawright and Scott T Young. A quality definition continuum. *Interfaces*, 26(3):107–113, 1996.

[100] Bhuricha Deen Sethanandha, Bart Massey, and William Jones. Managing open source contributions for software project sustainability. In *PICMET 2010 TECHNOLOGY MANAGEMENT FOR GLOBAL ECONOMIC GROWTH*, pages 1–9. IEEE, 2010.

[101] Maha Shaikh and Ola Henfridsson. Governing open source software through coordination processes. *Information and Organization*, 27(2):116–135, 2017.

[102] Helen Sharp, Nathan Baddoo, Sarah Beecham, Tracy Hall, and Hugh Robinson. Models of motivation in software engineering. *Information and software technology*, 51(1):219–233, 2009.

[103] Sharilyn Shiramizu and Amarjit Singh. Leadership to improve quality within an organization. *Leadership and Management in Engineering*, 7(4):129–140, 2007.

[104] Samuel A Silver, Rory McQuillan, Ziv Harel, Adam V Weizman, Alison Thomas, Gihad Nesrallah, Chaim M Bell, Christopher T Chan, and Glenn M Chertow. How to sustain change and support continuous quality improvement. *Clinical Journal of the American Society of Nephrology*, 11(5):916–924, 2016.

[105] Diomidis Spinellis and Vaggelis Giannikas. Organizational adoption of open source software. *Journal of Systems and Software*, 85(3):666–682, 2012.

[106] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. Code quality analysis in open source software development. *Information systems journal*, 12(1):43–60, 2002.

[107] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. Almost there: A study on quasi-contributors in open-source software projects. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 256–266. IEEE, 2018.

[108] Katherine J Stewart and Sanjay Gosain. The impact of ideology on effectiveness in open source software development teams. *Mis Quarterly*, pages 291–314, 2006.

[109] Klaas-Jan Stol, Muhammad Ali Babar, Paris Avgeriou, and Brian Fitzgerald. A comparative study of challenges in integrating open source software and inner source software. *Information and Software Technology*, 53(12):1319–1336, 2011.

[110] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering*, 24(3):1369–1417, 2019.

[111] Juan J Tarí and Vicente Sabater. Human aspects in a quality management context and their effects on performance. *The International Journal of Human Resource Management*, 17(3):484–503, 2006.

[112] E Ted Prince. Human factors in quality assurance. *Information systems management*, 10(3):78–80, 1993.

[113] Dave Thomas and Andy Hunt. Open source ecosystems. *IEEE Software*, 21(4):89–91, 2004.

[114] Linus Torvalds and David Diamond. Why open source makes sense. *Educause Review*, 36(6):70–75, 2001.

[115] William MK Trochim and James P Donnelly. *Research methods knowledge base*, volume 2. Atomic Dog Publishing Cincinnati, OH, 2001.

[116] June M Verner, Muhammad Ali Babar, Narciso Cerpa, Tracy Hall, and Sarah Beecham. Factors that motivate software engineering teams: A four country empirical study. *Journal of Systems and Software*, 92:115–127, 2014.

[117] Jon Welty Peachey, Alexis Lyras, Adam Cohen, Jennifer E Bruening, and George B Cunningham. Exploring the motives and retention factors of sport-for-development volunteers. *Nonprofit and Voluntary Sector Quarterly*, 43(6):1052–1069, 2014.

[118] Joel West. How open is open enough?: Melding proprietary and open source platform strategies. *Research policy*, 32(7), 2003.

[119] Minghui Zhou and Audris Mockus. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2014.

[120] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 350–361, 2019.

Appendices

A

Appendix A: Paper A

Influencers of Quality Assurance in an Open Source Community*

Adam Alami
IT University of Copenhagen

Yvonne Dittrich
IT University of Copenhagen

Andrzej Wąsowski
IT University of Copenhagen

ABSTRACT

ROS (Robot Operating System) is an open source community in robotics that is developing standard robotics operating system facilities such as hardware abstraction, low-level device control, communication middleware, and a wide range of software components for robotics functionality. This paper studies the quality assurance practices of the ROS community. We use qualitative methods to understand how ideology, priorities of the community, culture, sustainability, complexity, and adaptability of the community affect the implementation of quality assurance practices. Our analysis suggests that software engineering practices require social and cultural alignment and adaptation to the community particularities to achieve seamless implementation in open source environments. This alignment should be incorporated into the design and implementation of quality assurance practices in open source communities.

KEYWORDS

Open Source Software, Quality Assurance, OSS Community.

ACM Reference format:

A. Alami, Y. Dittrich, and A. Wasowski. 2018. Influencers of Quality Assurance Practices in an Open Source Community. In *Proceedings of 11th International Workshop on Cooperative and Human Aspects of Software Engineering, Gothenburg, Sweden, May 2018 (CHASE 2018).* https://doi.org/10.1145/3195836.3195853

1 INTRODUCTION

Open Source Software (OSS) communities have become a serious contender for commercial software supply. The open source paradigm is gaining momentum in strength and is increasingly adopted by the traditional software industry [1-2]. This industrial interest brings its own requirement for OSS, especially regarding quality. Traditional organizations use a combination of practices, processes, and techniques to produce quality software. Yet what can be done and achieved in a traditional setting might not be reproducible in an open source community. Hence, understanding the challenges of quality implementation in OSS communities is timely.

Little is known of how OSS communities perceive quality and the challenges of implementing quality practices in OSS communities. Community-based organizations are culturally different and have been established based on fundamentally different sets of values and goals. Software engineering practices and techniques seem to be designed generically to fit most circumstances and organizational settings. However, OSS communities have shown us that they can build highly professional products using social–technical processes [3, 4]. Although some OSS practices are inspired from the software engineering knowledge and practices, in most instances, the adoption deviates from the prescribed conduct. Scacchi [3] observes some "informalisms" in the adoption process that reflect the peculiarities of the involved community. He believes "informalisms" captures the uniqueness of how the community works and produces software. We want to learn from these particularities to assist in the implementation and adaptation of quality assurance practices.

The central objective of this study is to investigate how the community's social and cultural traits influence the implementation of quality assurance (QA) practices, techniques, and tools. We investigate the following questions:

RQ1: What are the forces influencing the implementation of quality assurance practices in the ROS community?

RQ2: How do social and cultural variables influence the implementation and execution of practices?

We define the term "practice" in line with as "... a common way of acting, acknowledged by a community as the correct way to do things. It can be taught to newcomers by letting them take part in this practice as an apprentice [5]. A community maintains the common practice through more or less formal 'articulation work' [6] which is also the means to handle exceptional situations. Ad-hoc behavior—always necessary to handle exceptions and to maintain the 'normal' [7]—is as such only perceivable by its deviation from both the formalized rules and the established practice." [8].

The ROS Community is large and diverse. Its Wiki platform receives over 1.4 million unique visitors a year and has 6,749 registered users. The community "discourse" receives an average of 150 posts a week. The total downloads of the .deb packages is over 13.4 million. Over ten years, ROS has become one of robotics'

CHASE'18, May 27, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5725-8/18/05...$15.00
https://doi.org/10.1145/3195836.3195853

de facto standard operating systems. The ROS community has different attributes than the commonly studied ones (i.e., Linux and Mozilla). First, it produces software components for robotics. Second, it is a multidisciplinary community. Third, most ROS developers are not software engineers. Their educational background is diverse but mainly from mechanics and electronics disciplines. In addition, members have varying professional experiences (i.e. packages developers, students, CTOs, etc.)

We find that the community has a strong quality awareness. Some industry-wide accepted practices have been implemented: (1) well-defined development process, (2) defects management process and tool, (3) code review, (4) continuous integration, (5) unit testing, and (6) knowledge sharing. However, these quality practices are experiencing implementation and execution challenges. Six forces and constraints—participation motives, priorities of the community, meritocratic culture, sustainability, complexity and adaptability of the community—have greatly influenced the implementation of QA in the ROS community. The cultural traits of the community also sway the QA implementation. Furthermore, the quality practices in place are constrained by sustainability issues and the complexity of the process of developing robotic systems.

This paper reports on a qualitative research on software QA practices following a mixed research method. Mixed methods deepen understanding of the research problem. Three techniques have been used: interviews with ten participants, virtual ethnography, and community reach-outs. In the last decade, the focus of OSS studies has been on "high profile" communities, like Mozilla, the Linux Kernel, and Apache. We selected ROS for its uniqueness as robotics software with a large and diverse participant base. This will contribute to the diversity of the samples studied previously.

2 RELATED WORK

We identified two streams of related work: (1) what the QA practices in OSS communities are and (2) how quality is assured in OSS development.

QA Practices in OSS Communities. ISO defines QA as "focused on providing confidence that quality requirements will be fulfilled" [9]. It is a set of activities for ensuring quality in software engineering processes that ultimately should result in quality software products. Halloran and Scherlis [10] surveyed eleven OSS projects to identify the QA practices adopted by the communities. They observed a variation in the adoption and implementation of QA. While some communities (i.e., Mozilla and NetBeans) have dedicated QA teams and well-established QA practices, other communities seem to follow the practices only rudimentarily. Unfortunately, no clear pattern emerges on when a practice succeeds and when it fails. In order to understand why some communities succeeded in implementing some QA practices successfully while others did not, we study the influencers of a successful QA implementation and execution.

Michlmayr and coauthors [11] studied quality practices in seven OSS communities. They found that the degree of adoption of quality practices influences the overall quality of the community product. They identify a set of quality practices taking place in the studied communities: new members joining, release management, branch management, peer review, testing, defects management, and standards and guidelines documentation. Although these practices are praised industry-wide, some are not formalized, and the implementation faces challenges. They list six quality practices issues: unmaintained code, managing variability, latency in security fixes and updates, ambiguous bugs reporting process, difficulty attracting new participants, and task coordination problems. However, they do not analyze the root causes of these issues.

Zhao and Elbaum [12] report that software development tools are popular in OSS communities; 75% of the respondents of their survey use configuration management tools, and 61% of the projects employ bug tracking tools. At the same time, documentation is not popular; only 32% of surveyed projects have design documents, and only 20% have documents to plan releases. More than half (58%) of the projects spent more than 20% of their time on testing, but only 15% of the projects spent more than 40% of their time on testing. It seems that larger projects tend to spend less time on their testing phase compared to smaller projects. They find that 20–40% of bugs are identified by end users. This agrees with a qualitative survey of Halloran and Scherlis [10]. The implementation of testing practices varies across communities. Also, non-programing activities (i.e., documentation) are not favored by contributors. Yet what makes some communities keenly embrace testing practices while others seem to shy from it has not been investigated. There is a need for studies to investigate further why non-programing tasks are not stimulating for OSS developers.

Rigby et al. [13] argue that despite being difficult to implement, code peer review is largely adopted by OSS communities as a central quality control practice. They studied the efficiency and the effectiveness of the practice in OSS communities. They found that the efficiency and the effectiveness of the practice depend on the level of the participation in the review process, the size of the change, and the author's experience and expertise. While the author expertise shortens the review cycle, the size and the complexity of the change elongates the cycle.

Lussier [14] recounts the experience of his team joining the Wine project, an open source implementation of the Windows API. The team is collocated and affiliated with a software strategy and research consulting services firm. Their first contribution did not pass the code review process since it did not meet the community standards. He recalls, "Team members watch the code carefully. No one wants to see bugs introduced into the source tree ... a real sense of ownership, of pride in the work, exists on Wine." Initially, the team resented the rejection. However, after three consecutive rejections, they adapted their programing standards to the community standards and conventions. This narrative is consistent with the widely accepted assumption that OSS quality is owed to peer review. Elsewhere, code is also claimed to be of high quality because it is created with passion,

and developers are highly motivated because they enjoy what they do [15].

How is quality assured in OSS development? In the closed source software development, QA relies on procedural rigor, extensive testing, and high testing coverage. Quality in OSS is assured by development (code modularity and frequent releases), not by control. Assuring quality is highly dependent on high participation in the project. High participation and frequent releases create energy in the development process. This facilitates bug discovery and generates a fast feedback cycle. Consequently, defects are identified and corrected more quickly [15].

Most successful OSS projects are sustainable and apply structured and organized development processes. Otte and coauthors [15] suggest that attracting talented contributors with diverse skills, implementation of a QA knowledge sharing infrastructure, standards and guidelines, and tools help to ensure sustainable quality practices. However, sustainability remains the key parameter for achieving quality in OSS communities [16, 17].

Open source software development necessitates specific methods and techniques for assuring quality [18]. Wahyudin et al. [19] claim that quality in the OSS development process is achieved via sustainability, peer review, and code modularity. They argue that code modularity enhances features evolution and minimizes bugs' introduction in the evolution process. Aberdour [17] believes that irrespective of the community dedication to quality, having a sustainability strategy is detrimental to assuring quality. Khanjani and Sulaiman [18] believe the discovery of bugs is dependent on the size of the community. The larger the community is, the more chances there are of bugs being identified, reported, and fixed, also known as Linus's law: "enough eyeballs, all bugs are shallow" [18]. In addition, having a knowledge sharing and collaboration platform facilitates knowledge dissemination and subsequently nurtures quality contributions and effective communication [19-20].

Abdou et al. [21] investigated some high-profile OSS communities' (Apache, Mozilla, and NetBeans) software testing practices and how they conform or deviate from ISO/IEC standards. The studied communities have matured testing practices. Still, the implementation of testing practices deviates from the prescribed industry version. Their study is limited to four large, successful, and well-established communities (i.e., Mozilla, Apache, NetBeans, and IDE).

Most of the identified work explored QA in isolation of other research streams, such as participation motives, culture, community sustainability, and community development. Practices execution cannot be completely detached from their milieu (i.e., organization, community) and the social context. They have to be studied in relation to the social, cultural, and organizational context of their milieu.

There has been a recent shift of interest toward QA in the context of open source software development [14-15]. It seems to indicate that quality is assured via the combination of one or more of these variables: code reviews, dedicated QA team, Linus's law, and a sustainability strategy. Simultaneously, other studies [6-8] appear to agree that software engineering QA practices make their way to OSS communities. There are significant empirical evidences that OSS communities adopt QA practices from software engineering. However, it seems that in some instances, the implementation of these practices is experiencing challenges [11, 12]. We cannot understand the success of software engineering practices in OSS communities without understanding how the social fabric and cultural variables correlate with these practices. The dependencies and the correlation between OSS community social and cultural variables and QA is not well-understood. Hence, this study suggests investigating what influences the implementation and the execution of QA practices in OSS communities. This is an initial step in a three-year research project to propose an implementation strategy for software engineering practices in OSS communities.

3 RESEARCH METHOD

This paper presents results of qualitative research in one case study community (ROS). The data was gathered using a combination of techniques: in-depth semi-structured interviews with ten participants, virtual ethnography, and community reach-outs. Qualitative research methods entail a structured process for the collection, organization, and interpretation of textual material derived from conversations, interviews, or observation [22-24]. One researcher spent 120 hours studying the community online infrastructure, forums, and virtual interactions. The researchers attended four community events. These community reach-outs were an opportunity to observe, be part of conversations, and experience the community atmosphere. Field notes were used to capture this exposure to the community. This ethnographic experience was complemented by in-depth interviews with ten active community members. Their professional roles and participation in the community varied from core developers to passive as users of the community code.

The data analysis was achieved by open coding, focus coding, and theoretical coding [24]. This study did not use grounded theory as an underlying research methodology, but its grounded approach has been the guiding process for the empirical data analysis.

Subject. ROS and ROS Industrial are the community subjects of this study. The Robot Operating System (ROS) is a middleware framework that is widely used in robotics. ROS provides standard operating system facilities such as hardware abstraction, low-level device control, and commonly used robotics functionality.

The underlying philosophy of ROS is to make universal software portable to different robotics systems. ROS is based on the concept of reuse and open source software. Its origins can be traced back to 2007. The project was incepted by the Stanford Artificial Intelligence Laboratory [25]. In 2008 a startup, Willow Garage, inherited the project. Five years later, in 2013, Willow Garage was absorbed by another company, and ROS "stewardship" transitioned to the Open Source Robotics Foundation [25]. Today, ROS is the de facto operating system for robotics.

ROS Industrial is "an open-source project that extends the advanced capabilities of ROS software to manufacturing" [26]. ROS Industrial is a branch of ROS with a specific industrial

application focus. Incepted in 2012, ROS Industrial has secured the collaboration of key players in the robotics industry (e.g., ABB, Yaskawa, Siemens, John Deere, BMW, Bosch, etc.). ROS Industrial's ambition is to become the worldwide open source standard for industrial robots.

4 FINDINGS

In response to RQ1 and RQ2, we observe that QA practices in the ROS community are influenced and constrained by the following forces:

1. Participation motives
2. Priorities of the community
3. Meritocratic culture
4. Sustainability
5. Complexity
6. Adaptability

The ROS community retains some of the software engineering and industry practices and processes. Many of the community QA activities still evolve; many experience challenges in the implementation and execution. Some of these issues are of a mechanical nature (i.e., outdated documentations) and require straightforward mechanical adjustment. However, a significant segment of the issues involves cultural alignment and/or alignment with the particularities of open source software development practices and processes. These issues manifest the cultural and social nature of the community. Addressing them will necessitate aligning the practice or the process with the cultural setting of the community.

These issues are merely a manifestation of unfit practices and adaptation failure to the environment ROS community. If there is a problem, then there might be reasons for it to exist in the first place. The issues exist because of a reaction to the introduction of change. The implementation of these practices did not cater to the social and cultural particularities of the community. It assumed that a default implementation will fit the community.

In the following, we are discussing the identified influencers by presenting the grounding for them in our data and in the existing literature. In the final paragraph of each subsection discussing an influencing force, we allow ourselves to speculate in what way it could be used to improve QA practices in this community.

4.1 Participation Motives

The participants do not consciously demonstrate the motives and their impact on their engagement in the community. Our analysis demonstrates that both intrinsic and extrinsic motives have influenced the implementation of QA practices in the community, mainly ideology and enjoyment. Some participants have strong ideological grounds, and it is manifested in their conduct and engagement in the community. Programming and the challenge of complexity are sources of enjoyment for some participants.

Intrinsic motivation refers to behavior that is driven by internal rewards [27] based on internal satisfaction and self-enjoyment [28]. The motivation to engage in an intrinsic behavior arises from within the individual because it is naturally satisfying. This contrasts with extrinsic motivation, which involves engaging in a behavior in order to earn external rewards [27, 28] that arise outside of the individual. It can involve tangible or psychological rewards. Psychological forms of extrinsic motivation can include praise and public acclaim [27, 28].

4.1.1 Ideology

INFLUENCER 1: OSS ideology is present in the ROS community thinking and decision making.

Data. Openness is the ideology attriute that influences practices in the ROS community. Some community members value this norm highly. In one of the community events (ROSCon 2017) that we attended, members discussed Slack as an online communication tool adopted by a group of developers for discussions and collaboration. Several community members refused to use it, while others were happy to continue using it. One member got emotional when the item came up for discussion and asserted, "I refuse to use it. It is not open source!" Another community member joined the opposition: "It is disappointing to see some people using a closed source, but I refuse to use it." There was an awkward silence before the discussion advanced to another subject. Apparently, not all community members rank openness equally high. Some have a relaxed and pragmatic attitude toward adopting closed source infrastructure and tooling.

Analysis. Openness is a mandated community requirement rooted in some members' ideological standpoints. Although adherence to the ideological beliefs are not shared with the same enthusiasm across the community, it is a fundamental variable and should not be dismissed. Similar to the Slack case, the implementation of a tool, process, or practice that does not embrace openness and transparency would create division in the community, and the chances of the practice being abandoned by a segment of the community members is high.

Openness is a manifestation of two cultural traits of open source communities: transparency and truth [4]. Pavlicek [29] believes that truth is a fundamental community asset. He explains that truth and transparency empower the community to produce "free software." Elliot and Scacchi [4] explain that "speaking the truth" is evident in the community social life and work practices.

"Ideologies are the shared framework of mental models that groups of individuals possess that provide both an interpretation of the environment and a prescription as to how that environment should be structured" [30]. The OSS ideology origins are deeply established in the "Free Software Movement" of the '80s led by Richard Stallman [4, 31, 32]. The movement is widely accredited for paving the way for the open source development.

There has been considerable interest in understanding the ideological framework of open source software communities [4, 15, 33, 35]. In settings such as OSS communities, where entry barriers are nonexistent and institutionalization of control is a challenge, ideology seems to facilitate order [34]. However, Ljungberg [34] suggests that commitment to the ideology varies widely across developers.

Impact. Stewart and Gosain [35] found that open source participants adhere to this ideology. David et al. [36], David and

Shapiro [37], Ghosh [38], and Ghosh et al.'s [39] surveys suggest that members' participation motives have underpinning ideological beliefs. OSS ideology needs to be analyzed, and its underlying beliefs and norms must be acknowledged before the design and the implementation of software engineering practices in OSS communities. We argue that ideology should be assimilated into the community practices to ensure its success. For example, when selecting a tool for a particular process, one should consider an open source to accommodate the openness feature of the community ideological values. A closed source tool will create division, and eventually the associated tool and practice will be abandoned.

4.1.2 Enjoyment

INFLUENCER 2: Enjoyment is equated with challenge in the community culture, while the QA tasks are not viewed as challenging.

Data. We observed that enjoyment is a key driver in community participation. Non-programing tasks are either being duly executed or taking place loosely in the community practices. An attendee at the community yearly conference commented on a poster that advocated an effort to implement and promote QA practices in the community: "What you are trying to achieve is formal; we developers seek fun in writing code and creating new features." "Skipping and skimming through pull requests and during code review are common occurrences," one participant stated. This attitude has its consequences. He further explained, "Things don't get reviewed and don't get the necessary attention for a longer time."

Analysis. Enjoyment as an intrinsic motivation has been associated with programming ("coding"). However, programming is not the only activity taking place in an open source software development environment. Other inherent tasks include code reviews, release management, documentation writing and maintenance, etc. There is a need to understand the relationship between non-programming tasks and enjoyment. In a community based largely on volunteers, this is a condition *sine qua non* for making software development processes function beyond programing.

Deci and Ryan's [40] self-determination theory is a widely supported contemporary intrinsic motivation theory. It suggests that humans have three intuitive psychological needs: a need to feel competent, a need to belong, and a need to feel independent. Intrinsic motivations emerge in people's behavior to support these psychological needs. Deci and Ryan [40] explain that when people feel competent, autonomous, and self-determined, they will seek to fulfill their internal self-satisfaction. Freedom of choice, the presence of a challenge, and the ability to overcome the challenge are the three variables that, when met, stimulate intrinsic motivation [40]. Non-coding tasks are not "challenging" or at least are perceived as not being so. They attract fewer contributors, and consequently, QA practices in the community (i.e., maintenance and testing) receive less attention.

Impact. It has been suggested that enjoyment is a key construct for understanding and explaining the motivation of OSS participants [27, 41, 42]. Lakhani and Wolf [27] suggest that enjoyment is a prevalent motivation amongst OSS contributors. Hence, it's not a force to ignore. Then what impact does this have on the implementation of QA and other non-programming tasks in OSS communities? It appears that one needs to immerse fun into non-coding tasks. This can be done by reframing the tasks or the process. For example, developing automated tests (that are programmed and leave credit in code repositories) may be perceived as more fun than writing manual tests. That would make it more challenging and more akin to programming.

4.2 Priorities Of The Community

INFLUENCER 3: QA is not high in the priorities of the community. Consequently, QA tasks are neglected.

Data. Priorities of the community are determined by the order of importance between various contributions and initiatives. Priorities are subject to change with changes in the community or with changes in people's objectives, motives, or knowledge. Innovation and functional depth and breadth are the priorities of the ROS community. "Everybody is aiming for new things," one participant stated. They thrive on innovating and resolving challenging and complex technical issues. This has been observed at the community yearly conference (ROSCon 2017). The main program was dedicated to new innovative features and use cases running on the community technological platform. One interviewee stated, "More importantly, our focus is features and functionality. The process is not always the priority." Another participant confirmed, "We want to also focus on the new stuff." In addition, new features are commonly announced and showcased in the community forum (i.e. Monocular Camera, and New packages for Lunar).

Analysis. Consequently, quality practices and continuous improvement are under-prioritized. A participant commented on the current QA processes in the community, "It takes a lot of time to set things up properly, and a lot of people see that as wasted time because you are developing a new component that is doing something. You want to focus on developing your component; you don't want to focus on setting up tests, gathering data, putting [out] a simulation, [and] all this kind of collateral work."

Impact. To counter this de-prioritization of QA, the OSS core team model could possibly be replicated for QA. In the core team model, a community sets up a dedicated team guarding and enhancing the core modules of the project. The core team model has been successful in the ROS community. This model could possibly be replicated for quality by creating a dedicated team to own and guard QA practices in the community. This would elevate priority rank of quality assurance to the level similar to programming core modules.

4.3 Meritocratic Culture

INFLUENCER 4: ROS culture of meritocracy is not integrated into QA practices.

Data. The cultural traits of open source software communities are grounded in the ideological beliefs and members' motivations. The study of the ROS community indicates that status attainment,

openness, freedom of choice, and the strive to innovate constitute the cultural traits of the community. However, meritocracy is a significant attribute of ROS culture.

Unfortunately, we observed that the community's cultural variables are not crafted into the implementation of QA practices. Fame and reputation are the rewards for those with superior technical knowledge, and they generously help others to resolve their technical challenges in the community forums. There is no such visibility or reward given to those who perform testing activities or documentation.

If enjoyment is one of the participation drivers, then quality should be fun. However, this is not the case. Fun is not constructed into QA practices. Fun is also the intellectual stimulation and challenge. Quality practices are conformance to rules, standards, and processes. Consequently, QA activities do not attract contributions. One participants stated, "Maintenance! No one wants to do that. I mean I am saying I am not happy that this is actually a real problem."

Analysis. Culture stems from a Greek word "cultura," meaning "to tend, cultivate, till, educate or refine" [43]. Bennett [44] defines culture as a shared mental system that distinguishes the members of one group from another. Culture is transferred from one generation to another. Fellows & Liu [45] argued that culture is ever changing as generations add something new to the culture before passing it to the next generation.

In a social system where meritocracy dictates the social structure and technical knowledge is awarded by social merits to attain higher community status, quality practices become trivial. This becomes more problematic when the innovation's functional depth and breadth dominate the community's priorities. In a local community meeting, a highly regarded developer was introduced to the crowd as being "famous worldwide" for developing a feature that was enthusiastically appreciated. Apparently, his contribution was of exceptional technical complexity. Others had unsuccessfully attempted to deliver it previously. In his presentation, he stated, "When you contribute a new feature, why think about quality? Just develop it and put it out there." This attitude of features first and quality later shows that quality is not built into the cultural environment.

Impact. According to Crosby [46], quality should be crafted into its cultural environment. It has to be part of the organization fabric, not part of the fabric. Culture and practices should be in synergy with each other. We observed that when a practice is alienated from the community culture, its implementation and execution fail. QA practices should be aligned to the meritocracy system. Non-programming tasks, especially QA, should be rewarded "karmas" similar to answering community members' questions. The "karmas" system is a reward scheme whereby members are rewarded "karmas" (i.e., points) for helping to answer questions in the community forum. Members with high "karmas" are highly regarded in the community.

4.4 Sustainability

INFLUENCER 5: The absence of a working sustainability strategy puts constraints on the execution and the development of QA.

Data. A subject states: "[Sustainability's] always the problem because if you don't have the large exposure, the project does not have much chance to survive after it has been developed." Finding a balance between quality and stimulating growth through ongoing contributions has been a challenge for ROS. While the flow of new contributions is steady, the core team does not have the capacity to ensure their quality.

"The main challenge is basically time and resources." In addition, the absence of a working sustainability strategy has led to a resourcing issue in the software maintenance activities. The community attempted a few initiatives to attract new maintainers; however, these have been unsuccessful. One participant explained, "So it is good to get people in, but it's hard to get maintainers in, both of which will actually continuously spend some of their time triaging and contributing. That's a huge challenge, and we have not figured out a good way to get more people involved, and that, I think, is one of the biggest challenges for the project." Consequently, a high number of packages end up being orphans and unmaintained. This applies to non-code artifacts as well; some QA Wiki documentation has not been updated for years.

Analysis. A sustainable community is "one that is economically, environmentally, and socially healthy and resilient" [47]. Resilience transcends inception and the ability to produce a product but rather than the product's ability to evolve and continuously innovate and thrive. Failing to create a sustainable environment to support themselves, OSS communities usually vanish.

Impact. Sustainability is difficult to achieve; however, a project's ability to attract and retain development and user resources increase the possibility of sustainability [48]. Communities that fail to create a sustainable environment to support themselves vanish. The evolution of the community product relies on ongoing creative contributions. Hence, OSS communities need to design and implement a working sustainability strategy to support growth and innovation.

4.5 Complexity

INFLUENCER 6: The complexity of robotics systems adds challenges to the implementations of QA.

Data. An interviewed mechanical engineer defines quality as the robot functioning defect free. Simulanously, a software engineer is disappointed that quality practices are not adhering to software engineering standards.

Analysis. QA of robotic systems is a challenging endeavor. Robots are complex distributed systems, combining control, AI, concurrency and mobility. Their development is a complex interdisciplinary practice. Their life cycle varies from the traditional software. This complexity of robotics development is not reflected in the current implementation of QA processes in the ROS community.

Impact. The complexity and interdisciplinarity of robotics systems is inherent to the ROS community; it is unlikely that it can be exploited to provide better QA. We consider it as a force

that additionally complicates an implementation of a successful quality management strategy for ROS.

4.6 Adaptability

We consider the organic self-alignment of practices to the community's social and cultural characeteristics. An organic modification does not occur via staged implementation nor via a design change process, but arises informally. The community implicitly acknowledges non-conformance to industry practices. For instance, there is no formal requirements engineering process in place. Instead, requirements are collected from ideas and code contributions of the members.

The self-alignment of practices is not always successful. The difference between a community-based adaptation of practices and a standard change management project lies chiefly not in the change but in the way the change is developed and integrated into the community. An adaptation should rooted in the community context. It requires deep understanding of what the community is doing, why people participate, how did they cope with past and present changes, their informality and how an organic adaptation has developed in the past.

4.6.1 "Informlisms" Of Processes

INFLUENCER 7: The ROS community does not follow QA practices as prescribed. It prefers an organic development of practices.

Data. Processes in the ROS community tend to take an organic course to full implementation, informed by its own trial of a practice as opposed to a planned implementation that comes from a well-established change design process known to the software engineering researchers. Some of the community practices were intended to be trials, which have since become abiding. "Most of the current processes in place have been thought in flight," a participant states. Another one explains why the adherence to the code review is fluid: "There is no formal process for that, which is probably something we could improve on, but there is no fixed rule [on] what to check for."

Analysis. These voices from the ROS community are consistent with earlier analyses. According to Elliot and Scacchi [4], OSS projects are often managed informally. Scacchi [3] suggests that OSS practices *software "informalisms"*, by not adhering to the traditional engineering practice, standards, and rationale. Sometimes the informalisms are democratically agreed upon through a voting system [33]. Sometimes they emerge implicitly. For instance, the traditional code inspection is fundamentally different from OSS code review [13].

Impact. We learn from this that the implementation of QA should be organic and allow for "informalisms." Prescribing practices top-down does not work. The OSS communities prefer to reflect, deliberate, and democratically consult the wider membership before adopting a change. During this process, practice adaptation occurs. Action research regarding QA processes in OSS should definitely take this into account.

4.6.1 "Ease Of Use

INFLUENCER 8: The ROS community has an affinity for ease of use.

Data. One participant summarized this elegantly: "Prioritization should also look at how long the task would take versus how important it is—and how adaptable the community to the task. You cannot bring a game-changing thing. People would say this is too complicated; I'm not going to do it. It's open source, not everybody aiming for stability. Everybody is aiming for new things. So you want to make sure those people are not scared away with complicated processes of testing."

Analysis. The community definition of ease of use is "minimum annoyance" and an enjoyable user experience. This is in line with "effectiveness," "efficiency," and "satisfaction of use." Consequently, the community members expect that QA practices are effective and efficient. QA processes should not delay or constrain developers' dedication to innovation.

Impact. Tools and processes should facilitate innovation and not constrain the creativity and participation. Ease of use should be a factor in the implementation of QA practices, and tools in OSS communities (which is obviously a challenge).

5 CONCLUSIONS

The ROS community has adopted accepted QA practices, but it is struggling with their effective implementation and execution. We focused on what influences this implementation and execution (RQ1 and RQ2) in order to understand the variables that contribute to the success and the establishment of QA practices in an OSS community. The implementation and execution of QA practices in the ROS community appears to be influenced by social and cultural factors and is constrained by sustainability and complexity. This shapes the practices toward a community-tailored variation rather than following the traditionally prescribed software engineering recipes.

What does this tell us? The identified influencers should be weaved into the design and implementation of QA in OSS communities. This necessitates some ingenuity and boldness. In commercial software production, QA practices are prescribed and enforced by management. In the future, We aim to change the trajectory of the ROS community to prioritize QA practices higher and to execute them effectively without management in traditional sense.

External validity. ROS shares the cultural and social attributes of other OSS communities. Hence, the related influencers will likely be applicable to other communities as well. The complexity of robotics software may not be representative of many other OSS projects, but it clearly is for some.

Acknowledgments. Work partially supported by EU's H2020 programme under ROSIN project, grant agreement No. 732287. We thank the interviewees for their participation.

REFERENCES

[1] Ø. Hauge, C.-F. Sørensen, and R. Conradi, "Adoption of open source in the software industry," in *OSS* 2008.

[2] Nagy, D., Yassin, A. M., & Bhattacherjee, A. (2010). Organizational adoption of open source software: barriers and remedies. *Communications of the ACM*, *53*(3).

[3] W. Scacchi, "Understanding the requirements for developing open source software systems," *IEE Proceedings-Software*, 149(1), 2002.

[4] M. Elliott, W. Scacchi, *Free software: A case study of software development in a virtual organizational culture* ISR 2003.

[5] Wenger, E. *Communities of practice: Learning, meaning, and identity*. Cambridge University Press, 1998.
[6] E. M. Gerson, and S. L. Star, "Analyzing due process in the workplace". *ACM TOIS*, 4(3), 1986.
[7] L. A. Suchman, "Office procedure as practical action: models of work and system design". *ACM TOIS*, 1(4), 1983
[8] C. Hansson, Y. Dittrich, B. Gustafsson, and S. Zarnak, "How agile are industrial software development practices?". *JSS*, 79(9), 2006.
[9] *ISO 9000: Quality management systems — Fundamentals and vocabulary*. 2015.
[10] T. J. Halloran, W. L. Scherlis, "High quality and open source software practices," *Workshop on Open Source Soft. Eng.*, 2002.
[11] M. Michlmayr, F. Hunt, and D. Probert, "Quality practices and problems in free software projects," in *OSS*, 2005
[12] L. Zhao, S. Elbaum, "A survey on quality related activities in open source," *SIGSOFT Soft. Engineering Notes*, 25(3), 2000.
[13] P. C. Rigby, D. M. German, L. Cowen, and M. A. Storey, "Peer review on open-source software projects: Parameters, statistical models, and theory," *ACM TOSEM*, 23(4), 2014.
[14] S. Lussier, "New tricks: How open source changed the way my team works," *IEEE Software*, 21(1), 2004.
[15] E. S. Raymond, *The Cathedral & the Bazaar*. O'Reilly 2001.
[16] T. Otte, R. Moreton, H. D. Knoell, "Applied quality assurance methods under the open source development model," in *COMPSAC'08*.
[17] M. Aberdour, "Achieving quality in open-source software," *IEEE software*, 24(1), 2007.
[18] A. Khanjani, R. Sulaiman, "The process of quality assurance under open source software development," *Computers & Informatics 2011*.
[19] D. Wahyudin, A. Schatten, D. Winkler, and S. Biffl, "Aspects of software quality assurance in open source software projects: two case studies from apache project," in *EUROMICRO*, 2007.
[20] R. Stallman, "Transcript of Richard M. Stallman's speech," free software: Freedom and cooperation" NYU, 29 May 2001,"
[21] T. Abdou, P. Grogono, and P. Kamthan, "A conceptual framework for open source software test process," in *Computer Software and Applications Conference Workshops*. IEEE, 2012,
[22] K. Malterud, "Qualitative research: standards, challenges, and guidelines," *The Lancet*, vol. 358, no. 9280, 2001.
[23] A. Howson, "Qualitative research methods," *Research Starters: Sociology (Online Edition)*, 2010.
[24] K. Charmaz, "Premises, principles, and practices in qualitative research: Revisiting the foundations," *Qualitative Health Research*, 7(14), 2004.
[25] *Open Robotics*. https://osrfoundation.org/ [6-Dec-2017].
[26] http://rosindustrial.org/. [Accessed: 06-Dec-2017].
[27] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being." *American Psychologist*, 55(1), 2000.
[28] K. R. Lakhani, R. G. Wolf, et al., "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," *Perspectives on Free and Open Source Software*, vol. 1, 2005.
[29] R. Pavlicek and Foreword by R. Miller, *Embracing Insanity: Open Source Software Development*. Sams, 2000.
[30] A. T. Denzau and D. C. North, "Shared mental models: ideologies and institutions," *Kyklos*, vol. 47, no. 1, 1994.
[31] G. Von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, "Carrots and rainbows: motivation and social practice in open source software development." *MIS Quarterly*, 36(2), 2012.
[32] S. C. Özbek, "Introducing innovations into open source projects," Ph.D. dissertation, Freie Universität Berlin, 2011.
[33] M. Bergquist and J. Ljungberg, "The power of gifts: organizing social relationships in open source communities," *Information Systems Journal*, vol. 11, no. 4, 2001.
[34] J. Ljungberg, "Open source movements as a model for organising," *European Journal of IS*, 9(4), 2000.
[35] K. J. Stewart, S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *MIS Quarterly*, 2006.
[36] P. A. David, A. Waterman, and S. Arora, "FLOSS-US the free/libre/open source software survey for 2003," *Stanford Institute for Economic Policy Research*, 2003.
[37] P. A. David and J. S. Shapiro, "Community-based production of open-source software: What do we know about the developers who participate?" *Information Economics and Policy*, 20(4), 2008.
[38] R. A. Ghosh, "Understanding free software developers: Findings from the FLOSS study," *Perspectives on Free and Open Source Software*, 2005.
[39] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Free/libre and open source software: Survey and study," 2002.
[40 E. L. Deci and R. M. Ryan, "The general causality orientations scale: Self-determination in personality," *Journal of Research in Personality*, vol. 19, no. 2, 1985.
[41] B. Luthiger, C. Jungwirth, "The chase for OSS quality: The meaning of member roles, motivations, and business models," in *Emerging Free and Open Source Software Practices*. IGI, 2007
[42] K. Lakhani, E. Hippel "How open source software works: 'free' user-to-user assistance" *Research Policy* 32(6) 2003.
[43] L. Smircich, "Concepts of Culture and Organizational Analysis". *Administrative Science Quarterly*, vol. 28, no. 3, 1983.
[44] T. Bennett, "Cultural Studies and the Culture Concept". *Cultural Studies*, vol. 29, no. 4, 2015.
[45] R. Fellows, and A. M. Liu, "Use and misuse of the concept of culture". *Construction Management & Economics*, 31(5), 2013.
[46] B. Crosby Philip, "Quality without tears: The art of hassle-free management," 1984.
[47] *Institute for Sustainable Communities*. [Online]. Available: https://www.iscvt.org/. [Accessed: 16-Dec-2017].
[48] I. Chengalur-Smith, A. Sidorova, S. Daniel, "Sustainability of Free-Libre Open Source Software projects: A longitudinal study." *Journal of Association for Information Systems*. 11, 2010

B

Appendix B: Paper B

2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

Why Does Code Review Work for Open Source Software Communities?

Adam Alami
IT University of Copenhagen
Denmark

Marisa Leavitt Cohn
IT University of Copenhagen
Denmark

Andrzej Wasowski
IT University of Copenhagen
Denmark

***Abstract*—Open source software communities have demonstrated that they can produce high quality results. The overall success of peer code review, commonly used in open source projects, has likely contributed strongly to this success. Code review is an emotionally loaded practice, with public exposure of reputation and ample opportunities for conflict. We set off to ask why code review works for open source communities, despite this inherent challenge. We interviewed 21 open source contributors from four communities and participated in meetings of ROS community devoted to implementation of the code review process.**

It appears that the hacker ethic is a key reason behind the success of code review in FOSS communities. It is built around the ethic of passion and the ethic of caring. Furthermore, we observed that tasks of code review are performed with strong intrinsic motivation, supported by many non-material extrinsic motivation mechanisms, such as desire to learn, to grow reputation, or to improve one's positioning on the job market.

In the paper, we describe the study design, analyze the collected data and formulate 20 proposals for how what we know about hacker ethics and human and social aspects of code review, could be exploited to improve the effectiveness of the practice in software projects.

***Index Terms*—Open Source, Code Review, Motivation**

I. Introduction

Code review is an established software engineering practice, that ensures good quality of source code, lowers bug frequency, and enforces coding standards [1]. During code review, reviewers (software engineers other than the author) read code in order to point out mistakes, shortcomings, and convention violations that had been overlooked during programming. The practice has evolved over the years from simple inspections of sections of code to formalized techniques that give immediate feedback. Reviews are performed in various forms, such as pair programming, informal walkthroughs, and mandatory approvals before code merging. A variety of tools have been developed to help reviewers scrutinize and check the viability and functionality of code during these reviews, and to formulate feedback.

Code review is particularly successful and cherished in free and open source software (FOSS) communities [1]–[4]. Some would go as far as to say that code review is the *raison d'être* of FOSS: *"code review ... is the reason behind open source. If anyone could contribute to a project, there could be chaos."* It is a *"wall that separates bad code from good code."*[1] When asked about relative importance of code review and testing, code review is often ranked clearly above testing by FOSS engineers.

[1]Statements by FOSS engineers interviewed in this study.

The effectiveness of code review depends on the level of participation, the size of the changes made, and the reviewer experience and expertise. Thus it is not an entirely obvious practice to implement. Now, that code review is also widely used in the industry [4], [5], it is particularly relevant to understand why and how it works. As, the standard of peer reviewing in the open source environment remains a beacon of best practice, we turn our attention to FOSS projects for insight. We investigate the practice from the perspective of the main participants, their motives and behaviors. We ask:

RQ: *Why does code review work for FOSS communities?*

We want to understand how contributors deal with the inherent negative feedback; what motivates them; and what values lead them to first contribute code of high quality, then produce high quality feedback, and finally to diligently consider the feedback to improve the contributions. We ask this question to (i) learn from FOSS communities to translate the experience to closed-source environment, (ii) to help other projects in implementing the practice successfully. We formulate observations based on data and then speculate how project and community managers can incorporate these results into their work culture. We find that:

- FOSS contributors experience *rejection and negative feedback* regularly. Communities do not eliminate this negative experience, as this seems to be the core improvement mechanism of code review.
- Our subjects develop *mature attitude to negative feedback*, taking it as an opportunity to learn, to improve, and ultimately excel in their job.
- The *ethic of passion* drives the contributors. They are passionate about all aspects of the project, including the reviews. The passion allows them to invest themselves into code review, and it also makes them resilient to negative interactions.
- Community members develop a working *ethic of care*, showing commitment and care toward the project, the community, and other developers. The code review is a gate keeping practice, an implementation of the care for quality of the project. Thus care is a strong motivator for both for performing the code review, and for diligent execution.
- *Intrinsic motivation:* Altruism and enjoyment are key intrinsic motivators of FOSS code reviewers. Even

1558-1225/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE.2019.00111

paid-for code reviewers are effectively volunteers, who choose tasks following intrinsic interests.
- FOSS communities have developed a *gift economy*, centered around a range of non-monetary non-material *extrinsic motivators*: reciprocity of contributions, sharing success, displaying status and reputation, transfer of FOSS reputation to professional career, continuous learning and development, all the way to punishment for under-performance. These economy "pays" for code review effort.

In the final part of the paper, we reinterpret these observations to extract ideas for improving the code review process in existing projects and communities. We hypothesize that practices described by our subjects (for instance, communicating guidelines, or establishing mentorship) are the reasons behinds their thriving during code review. We thus formulate them as imperative suggestions, that, we hope, can be used for further investigation in action research projects.

We start by characterizing the studied communities in this paper, using them also to cast some light on the practice of FOSS code review (Section II). Section III describes the study design. Section IV is devoted to the analysis and interpretation of the data. We aggregate the actionable hypotheses in Section V, discuss related work in Section VI, and conclude in Section VII.

II. Subject Communities

We begin by characterizing the five studied FOSS communities and how do they use code review. Throughout the paper we use the terms developer, programmer, software engineer, and hacker interchangeably to refer to FOSS contributors.

Robot Operating System (ROS) is a popular open-source middleware for robotics. It provides standard communication and coordination features, and bundles implementations of essential robotics-specific functionality with software drivers for popular hardware components. A ROS-based application is created by complementing selected ROS components with application specific code, typically in Python or in C++. The project originated in 2007 at the Stanford Artificial Intelligence Laboratory. In 2008, it transferred to Willow Garage, a robotics start-up. Since 2013, it has been stewarded by the Open Robotics foundation.

Since its inception, the community enforced reviews for every pull request, even for changes from the core team. Common code inspection meetings were also organized. Over time the review practices have deteriorated and are abandoned due to lack of resources. This development partly motivates our study—we work with the community to re-energize the practice through an action research style intervention. As a result the community is initiating a pilot project using standard GitHub tools for code review and a policy that encourages code contributors submitting pull requests to review their peers' contributions.

Apache Allura is a hosting platform. Allura integrates key development tools (version control repositories, issue tracker, wikis, blogs, etc.) for developers. It is an open source project, under the umbrella of Apache foundation, implemented in Python. Since 2012 it is used by SourceForge as the main platform.

The official community code guidelines[2] require that each contribution is tested in a local fork prior to submission. Each pull request must be accompanied with the necessary test cases to be added to the automated test suite of the project. As a precondition to the submission, the contribution must pass all existing tests. Contributions with "open" status are reviewed and discussed in the Git Merge Request discussion forum or in the issue tracker. Violations of the community code review are highlighted, architectural and programming decisions are debated. Good contributions are praised. A contribution could be rejected for minor code guidelines violations. Any member of the community can review code; however, ultimately it is the decision of the maintainer whether to merge a contribution or not.

The Comprehensive Knowledge Archive Network (CKAN) is an open source project developing a web-based storage and distribution platform for data, mostly used by public institutions joining the open data movement. CKAN is implemented primarily in Python and JavaScript. The project is overseen by the Open Knowledge International association.

The CKAN community uses GitHub for pull requests management and code review. Each contribution must be accompanied with the relevant tests and updated documentation. Once a pull request is submitted, it is subject to review by one to four other community members. The community specifies coding standards for all used technologies (i.e. Python, CSS, etc.). Architectural and programming decisions are debated and minor code guideline violations are pointed out. The official review guideline recognizes that besides standards, reviewer's judgment is a key factor in accepting or rejecting contributions. [3]

FOSSASIA is a community centered in Southeastern Asia developing software for social change since 2009. FOSSASIA projects are not limited to software applications, but include also hardware and design.

FOSSASIA focuses "*on the code quality more than on managing pull request ethics*,"[4] emphasizing the actual goal of the code review work. Its best practices are documented in programing and commit style guidelines. Pull requests are reviewed using standard GitHub facilities. Up to eight reviewers participate in a single discussion. Acceptance requires reaching a consensus between reviewers. It is the responsibility of the core team to review code. Reviewers in this community can be pedantic, even grammatical errors in the code comments are pointed out. Still, FOSSASIA manages to maintain a friendly atmosphere. Reviewers offer help to fix errors if required.

The Linux Kernel is the most popular and versatile operating system kernel on the planet, used on super computers and web-servers, powering up cloud infrastructure, and controlling lots of mobile and embedded devices (including all Android devices).

[2] https://forge-allura.apache.org/p/allura/wiki/Contributing%20Code/
[3] http://docs.ckan.org/en/2.8/contributing/reviewing.html
[4] https://github.com/fossasia/susi_server

Since its inception in 1991, the project is a success story, especially in terms of developing a sustainable community.

Unlike other communities, Linux is not using GitHub for code review, but communicates changes and performs the review using mailing lists. The code style guideline and pull request submission documentation are thoroughly detailed. Even the size of the email is specified. Contributors are encouraged to prepare patches that are concise and logically atomic. Contributors submit their patches to the relevant subsystem mailing list, where they are reviewed by volunteers. Criticism and other comments are exchanged very openly in the mailing list. Code is pedantically reviewed; there are known cases when a patch went through 20 review iterations. Acceptance of a patch is subject to a community consensus, but the ultimate decision resides with the maintainer.

III. Method

Recall that we are asking *Why does code review work for FOSS communities?* To answer this question we want to dig deeply into the mindset of participants in this inherently human process that relies heavily on communication skills; that involves feedback, critique and rejection on daily basis; and that exposes power and decision hierarchies in communities. For this reason, we choose a qualitative research method that is suitable for exposing participants' experiences and motivations. We have conducted 21 semi-structured interviews with members of the communities that are using code review (Apache Allura, CKAN, FOSSASIA, Linux). To this we add observation of three meetings of the ROS community (10–16 participants) devoted to implementing a new code review process. While the interviews explain mostly the experience with the existing process, the meetings are more likely to review the expectations, wishes, and concerns with the process, as during the meetings people reflect about consequences of instituting it.

A. Interviews

While a structured interview has a rigorous set of questions, which does not allow to divert, a semi-structured interview is open, allowing new ideas to be brought on top of a predefined question framework. The flexibility is used to enhance the depth of interviewee's statements. This makes semi-structured interview suitable for capturing rich qualitative data and obtaining deep insights into people's believes and behaviors.

TABLE I
KEY PARTS OF THE INTERVIEW FRAMEWORK

intro	Can you talk to me about your community?
	What first motivated you to participate in this community?
core	Can you describe the code review process in your community?
probing	What makes you adhere to best practices?
	How do you cope with the feedback?
	What makes you want to participate in code review?
	Can you share with me an example of good feedback you received, a part of the code review, and how did you feel about it?
	Can you share with me an example of negative feedback you received, part of the code review, and how did you feel about it?

The questions in our interview framework fall into three categories: introductory, core, and probing (Tbl. I). The introductory questions were designed to warm up the conversation. The core question related directly to our research question. The probing questions aimed at making the conversation detailed and concrete. We also asked other questions (not in the table) about quality assurance practices in FOSS. These questions sometimes revealed the relation of code review to other practices, as shown in the statements reported in Sect. I.

B. ROS Community Meetings on Code Reviews

The interviews were executed with members of the communities that successfully use code review. In contrast, the ROS community has failed to sustain the practice in the past, and now, having grown substantially, it attempts to reboot it. As part of our action research involvement with the ROS community we facilitated community meetings, where QA practices are being discussed and implemented. Re-instituting the code review was one of the highest prioritized activities of this group (within the top three of the sixteen prioritized actions).

Three one hour long community meetings were dedicated to the implementation and the logistics required to implement a code review process in the community. The meetings had an open structure. Ideas to implement, and details of implementations were proposed and voted by participants (and not by the facilitators, the authors). It is important that the meeting group takes the execution of decisions on themselves, so they have to bear in mind the cost of the implementation. In the meetings, we also observe reflections on the prior attempt to implement code review in the community, and reflections on this. The meeting data complements the interview data in following ways: we collect opinions of much more contributors quickly, we can see how differing opinions are confronted, we see how ROS contributors discuss the failure of code review, and how they anticipate the practice in a community that is not yet converted to use it.

C. Subject Selection

We have selected the five FOSS communities (Sect.II), to achieve a deep understanding of the phenomena under study. We interviewed 21 participants from Allura, CKAN, FOSSASIA, and Linux communities. We searched for contributors on LinkedIn, using community name and terms "contributor"/"developer". We contacted random entries from the search results and used snowballing to increase the sample. We had no prior relationships to any of the subjects. Table II summarizes the demographics of the population. The role labels are self-selected by participants (on LinkedIn profiles). The majority of the participants contribute to FOSS as part of their professional employment and are paid for their contributions. Two of the participants were students. Twenty participants were males and one was a female.

The participants for ROS community meetings on quality assurance have been recruited using two methods: directly inviting community members to participate (eight individuals) and via an announcement in the community public forum (15 individuals). The group counts predominately developers, but also some directors, project managers and CTOs of companies using ROS are amongst the group members. Not all 23 participants joined all three meetings; attendance was in average 16 participants.

D. Data Collection

Due to geographical distribution of subjects, all interviews were conducted remotely, using Google Hangouts. Each interview lasted 40-60min and generated on average 14 pages of verbatim. The ROS QA group meetings used an electronic meeting platform, GoToMeeting. We transcribed the recorded audio, generating 16 pages of verbatim per meeting on average.

E. Analysis

We analyzed the material from interviews and meetings following the guidelines of Robson and McCartan [9] and of Miles and coauthors [10]. The analysis was iterative, started in early stages of data collection, and continued throughout the study. First, the open coding enabled us to retrieve and compare the text that has been linked to a particular theme. We devised the codes by examining the data line-by-line using the following questions as a lens: What is this saying? What does it represent? What is happening in here? What is she trying to convey? What is the process being described?

Then we searched for patterns in statements and ideas, formulating themes. A theme is a concept, an implied topic that organizes a group of repeating ideas that help to answer the study question [9]. We used *analytical memos* to formulate and work with the themes: the first author compiled a number of memos based on the coding that summarized and aggregated observations. These where used as discussion material between the authors. Table III captures the identified themes, examples of verbatim, and argues why a particular theme was selected.

TABLE II
DESCRIPTION OF THE STUDY POPULATION

Participant	Community	Role	Experience [Y]	Country
1	Allura	student	2	India
2	Allura	software developer	12	USA
3	Allura	senior software developer	14	USA
4	CKAN	software developer	10	Slovenia
5	CKAN	software engineer	12	UK
6	CKAN	student	2	Slovenia
7	CKAN	senior software engineer	8	UK
8	FOSSASIA	software developer	2	India
9	FOSSASIA	software engineer	8	India
10	FOSSASIA	software engineer	10	India
11	FOSSASIA	software developer	8	India
12	FOSSASIA	software developer	7	India
13	FOSSASIA	software engineer	13	India
14	Kernel	Linux kernel engineer	18	Denmark
15	Kernel	Linux kernel hacker	10	Denmark
16	Kernel	principal engineer	23	Brazil
17	Kernel	embedded Linux engineer	5	Spain
18	Kernel	embedded Linux engineer	7	USA
19	Kernel	Linux kernel engineer	10	USA
20	Kernel	Linux kernel engineer	12	USA
21	Kernel	senior project manager	30	USA

IV. FINDINGS

Code review is an emotionally loaded practice, with lots of exposure of reputation and ample opportunities for conflict. In the following we present our findings, which explain how the successful communities deal with these issues. In a nutshell, we stipulate that the underlying reason behind code review success in FOSS is hacker ethics. Himanen [11] argues that the hacker ethics are more about moral virtues, in contrast to the protestant work ethic, which stresses diligent hard effort. Hacker's values, according to Himanen include but are not limited to passion, caring, creativity and joy in creating software.

A. Rejections & Negative Feedback

A publicly communicated rejection of a contribution is a common experience in FOSS code review, across all studied communities. The reports from the Linux kernel community are most pronounced, where some even speak of *"rejection by default,"* assuming the rejection as the initial position (Participant 16). An examination of the Linux Kernel mailing list archives shows that the language used can be intimidating.[5] The Linux kernel community uses frequent rejections and the harsh language deliberately as a *"congestion control"* mechanism (Participant 13) that limits the overflow of contributions.

How do contributors handle rejections and negative feedback in code review? Given the vulnerability of contributors in the code review process, one would expect that FOSS communities would be decaying. Yet, the communities, we studied, are flourishing. It appears that ability to deal with rejection is *sine qua non* for succeeding in open source: *"for someone to succeed he needs to be able to handle rejections, rudeness, and jarring-to-the-senses language"* (Participant 14). In the remainder of this paper we investigate the mechanisms that we observed at work, that, among others, minimize the negative effects of receiving critical feedback: learning from rejection, the ethics of passion, the ethics of care, and reputation.

Implications. Crucially, none of the subject communities attempt to eliminate rejection and negative feedback from their development process. This is clearly not a route to deal with the issues of code review. Anybody implementing a code review process should institute an environment where rejections are common, accepted, and normal. Mentoring and training should be considered to support newcomers to the practice in learning how to handle rejections.

> ***Observation 1.*** *Contributors are subject to frequent rejections in code review. Communities neither reduce nor eliminate the negative feedback, as they believe it is core to the practice.*

[5] https://lkml.org/lkml/2018/8/3/621

TABLE III
THEMES: EXAMPLES, DEFINITIONS, AND WHY THEY WERE CHOSEN

Theme	Definition	The theme in our data	Example verbatim
Rejection	An action taken by someone of not accepting, trusting, or considering a contribution of another community member. In the context of code review, this refers plainly to the refusal to include the contributed code in the main project.	Rejection is an inherent and dominating characteristic of the code review process, thus it appears in our data naturally. Rejections were discussed frequently, both directly and indirectly.	*I was completely depressed. It was some feedback I got on Friday. I have a patch which has gone through 5 revisions ... I had these 5 revisions and this has not happened! ... I got these two guys which they are anti-social. They picked a piece of my code and say why do you do this? This is crap, it will hurt performance ... They never say something nice.* Participant 14
Iterative improvement	In code review, a cycle of repeated review, rejection, and improvement (in response to criticism) of the same contribution.	Iterative improvement (our name) was brought up by several participants, both in negative and positive sense. Some participant perceive it as "depressing" other see it as an opportunity to learn and grow.	*Only a certain type of people can handle this. It's not very healthy on the mental state. You have to be able to handle this and very persistent. The last patch I got in has gone through 7 revisions before it got in.* Participant 14.
Passion	A strong inclination toward a significant activity in one's life. Passion is often self-defining, pertinent to one's identity. It is a necessary component in reaching the highest level of achievement, and contributes to creativity. It affects autonomy, competence, and relatedness [6].	Passion occurs directly numerous times in data. Subjects also talk about their community and work passionately. They speak with certainty, in higher pitched and faster voice that demands attention, with positive and assertive body language.	*... down the line, you always get to be attached to the project and get the passion of contributing and getting it out to the world so yes, it is one of the reasons why people contribute.* Participant 8
Caring	A relationship where the "caring" person acts in response to a perceived need from the "cared-for." A caring relationship is a basic human instinct, a universal virtue. The caring party engages in helping the cared-for [7].	Caring repeatedly appears as a core value in the interviews and discussions, with symptomatic phrases like *"I care," "we care"*. We see both care for abstract entities (the project, community, quality) and for community members.	*In FOSSASIA we care. It's not like an average job. We do it because we want to do it and we care about the quality.* Participant 10
Intrinsic motivation	An internal desire to perform an activity. Self-applied. Arises from a direct relationship between the individual and the circumstances. The reward is intangible—a sense of achievement or satisfaction. Intrinsic behavior springs from the human need for competence and self-determination, directly derived from the emotions of interest and enjoyment [8].	While talking about their tasks, participants repeatedly used phrases like "makes me happy" or "feels good/nice."	*It feels nice doing something for the community. There is satisfaction, especially when the PR is merged. It feels nice!* Participant 10
Extrinsic motivation	Inspiration to act to gain some external reward [8], valued by goal-oriented individuals. An extrinsic reward is tangible or physically given to award one's participation. Extrinsic rewards are easier to exploit in project management than intrinsic ones.	This theme emerged as participants talked repeatedly about the significance of reputation to themselves or to their peers.	*It's a great piece of software and a success story. Everybody wants to be part of it. Not only that, having a reputation in the community also counts.* Participant 16

B. Iterative Improvement

Code review in the studied communities is iterative. Typically, after a rejection, or in response to negative feedback, the contributors implement necessary improvements, and ask for another review. A pull request may go through 20 iterations of review in the Linux community (Participant 15). In FOSSASIA, the range is 1–4 iterations (Participant 12). This amount of iterative scrutiny may appear intimidating at first. One could be tempted to conclude that the comfort of the contributors is sacrificed in the name of quality. However this is not the view shown by our subjects: Iterative improvement is a mechanism to turn the negative feedback into a positive experience—they can advance their technical excellence in the process.

How learning affects code review? Ghosh and coauthors observe that knowledge is a salient motive for participation in FOSS [12], [13]. Lakhani and Wolf report that 45% of the survey's participants join a FOSS community to improve their skills [14]. FOSS creates a positive environment for learning [13]. Our subjects concur. They use terms such as *"opportunity to learn"* (Participant 15) and *"self-growth"* (Participant 21) when referring to processing feedback. The rejection and negative feedback are rationalized from an issue to a reason for individuals to join the process.

> ***Observation 2:*** *The iterative improvement cycle in code review turns negative feedback into a positive opportunity for learning and technical-growth by contributors. Receiving feedback may even become a reason for participation.*

Implications. Reacting to feedback in mature ways can be learned. Organizations and FOSS communities should consider using, for instance, coaches and mentors to help the participants in code review to develop a constructive attitude to feedback.

C. Ethic of Passion

Passion is a strong inclination or desire toward an activity that one likes or even loves, that one finds important, and one invests time and energy in [15]. Passion is a necessary component in reaching the highest level of achievement, and contributes to creativity [15]. FOSS contributors review code

that does not concern them in any way out of their passion for programming, passion for the community's project, and passion for excellence. *"In open source or at least in the Linux Kernel community it's not a job like in a company, even though most people are getting paid now. People have passion about this. I think the most important thing is how they perceive the criticism when they are passionate about the work. I think we don't see it as criticism"* (Participant 20). Participants see passion also as a help in coping with negative feedback. They speak with passion and a sense of purpose.

How does passion shape the execution of code review? Vallerand et al. propose a dualistic model of passion, distinguishing obsessive and harmonious passion [15]. *Obsessive passion* is tied to a person's self-esteem, ego. The person's identity is defined by the passion, thus she is compelled to engage in the activity. Such person engages in an activity rigidly. Obsessive passion generates strong negative effect when the person is unable to be involved in the activity. Our data does not show any strong links between the activity of code review and the participants' identity, and we do not observe any negative effect regarding inability to perform code reviews. Hence, we conclude that passion of the studied subjects is of the *harmonious* type. Harmonious passion results from an autonomous internalization of acceptance of the activity [15]. The desire to participate in the activity is significant, but not overpowering. Harmonious passion generates greater positive effect than obsessive passion. Individuals subject to harmonious passion demonstrate better concentration, better flow, and flexible persistence.

According to Vallerand's model, passion should positively affect the quality of the performed code reviews, and our subjects concur, for instance: *"passion is the force behind the quality of the work and people contributions"* (Participant 11). Interestingly, Bonneville-Roussy et al. [16] analyze the types of goals that people subject to harmonious and obsessive passion set. They note that only harmonious passion generates so called mastery goals, which is consistent with our subjects being very passionate about technical excellence, and the participation in code reviews to achieve the excellence (see Observation 2).

> ***Observation 3:*** *The ethic of passion motivates FOSS contributors. Consequently, they dedicate effort to code review, deliver high quality, and are more resilient to rejection.*

Implications. It is definitely difficult to operationalize passion. Passion is innate, but it can be developed and nurtured like any other value. It should be nurtured in software engineering environments. If code review is important for an organization, passion for the practice should clearly be key for selecting the project members (as opposed to training skeptics to perform code review). Since passion is contagious, organizations and projects should strive to recruit passionate managers and team members, and encourage them not to be shy about their passion.

D. Ethic of Caring

Toombs et al. [17] argue that hacker communities demonstrate a nonliberal ethos, prizing self-determination, technological expertise, independence, freedom from government, and suspicion of authorities. However, for these communities to function, care values are also important, those values of collaboration, cooperation, and support of one another. For FOSS contributors, work is about the power of human reunion, about working together and caring for each other, for the community, and for the project. *"People in the community care about the work. They care about the community, about the product and its quality. Everybody cares. This caring together makes a difference. You don't feel [like if you were] working for a company."* (Participant 11) Caring, as articulated by the subjects, is the feeling and the display of concern and attaching importance to the community work and its products.

How does care shape the execution of code review? Subjects believe in a positive correlation between caring ethic, work satisfaction and performance. Community members appear to naturally care for the project and do not want anything that would hamper its effectiveness and efficiency. Code review is a way to execute that care for reviewers, to enforce the quality requirements. Asked how people cope with a heavy review load in the Linux community, Participant 16 stated *"People care about quality in this community. Not only that. People are passionate about the project."* Participant 3 puts code reviews and care as the two most important factors behind the success of FOSS: *"number one is peer review, so that is one of the main practices. Number two, we try to do it with care. Number three, we try to have as many tests as possible and we try to have at most 80-90% code coverage."*

Paradoxically, caring is the motivating factor for some subjects, also when they are reviewees. Care shown by others may help to deal with negative aspects of code review. Participant 1 names the caring attitude of his community mentors as the main motivator behind his persistence in the early days. Others talk about reviewers who not only criticize contributions, but also offer help to improve them.

> ***Observation 4:*** *The ethic of care drives our subjects. They use the gate of code review to exercise care for quality. Care also helps them to control the negative feedback.*

Implications. Care is easier to habituate than passion. Thus companies and projects should find it easier to exploit it, by instituting an ethic of care. A caring attitude can be rewarded and encouraged. It appears that a successful implementation of code review would be helped if leaders cared about how contributors cope with feedback, and if they themselves shown care for code review. Note that in many FOSS communities the project leader is actually the most active code reviewer.

E. Intrinsic Motivation

Intrinsic refers to the innate, the natural, the part of a whole that cannot be removed from the whole nor the whole from it. Intrinsic motivation derives from internal satisfaction. Some subjects directly name natural affinity for programming behind their dedication to work (including code review), adherence to best practice and assuring the overall quality of the tasks.

Altruism and enjoyment were observed to be the main intrinsic motivation amongst our participants, for instance: *"The*

TABLE IV
EXTRINSIC MOTIVATORS IN THE STUDIED COMMUNITIES

Motivator	Description
Reciprocity	Known in the literature as the gift economy. Other community members respond positively to your contributions, through mechanism like reviewing code for each other, offering help, public appreciation, etc.
Participation in success	Success is attractive, and contributors find it rewarding to be part of a successful project, and being able to help, or given responsible roles in it (such as code reviewer).
Status, reputation	Work experience in a FOSS project accumulates technical expertise and social capital elevating the contributor status. Since the management hierarchy is much less important than in commercial organizations, the status matters more here. Reputation and status might be intangible, or expressed using a metric system (like Karma points, forks, likes, etc.)
Career building	Both intangible and "tangible" reputation accumulated in successful open source projects, translate into carrier opportunities for engineers: interesting job offers, presenting at conferences, etc.
Learning	Code review, like any feedback process, provides peer-learning and development opportunities. The reviewers act like masters in relationship to apprentices (contributors). These roles can of course swap for the same individuals.
Be amongst the best	Code-review introduces a high acceptance bar, deterring mediocre submissions. Contributors' apprehension of critique is an extrinsic motivator that leads to better initial submissions and more diligent improvement to feedback. Unlike the previous five, this is the only negative extrinsic motivator (exercised by code reviewers) that we have seen in our study.

feeling that your code is going to be used, maybe, for future. Maybe its going to help some people and that's something I really like" (Participant 7). The FOSS motivation literature suggest that enjoyment is a key motivator for contributors [14], [18], [19]. Subjects are *"happy"* (Participant 7) or experience *"nice feelings"* (Participant 21) when performing code review. *How does intrinsic motivation influence the execution of code review?* Intrinsically motivated employees perform well, behave effectively, and remain loyal to the organization [20]. Rogstadius and coauthors determined that increased pay does increase worker's willingness to accept a task and faster completion, but pay does not affect the quality of the work [21]. They claim that, intrinsic motivators can lead to higher quality work, in fact, higher than extrinsic rewards.

Our subjects agree with these findings, Participant 11 states *"It's a great feeling. I don't know how to describe it, but feels nice. It drives me always to do more and better."* The intrinsic quality in the execution of code review starts from the selection process. Contributors voluntarily select the patch they are comfortable to review. This is different from many closed-source environments, where reviewers are assigned to review code.

> ***Observation 5:*** *Altruism, and enjoyment are key intrinsic motivators for our subjects. Open source reviewers are effectively volunteers (even if paid) and can choose review tasks following intrinsic interests.*

Implications. The very nature of intrinsic motivation is that it cannot be easily controlled externally. Perhaps, the only way to exploit it, is to watch for the symptoms (altruistic behaviours, enjoyment), and take them as indicators of good code reviewers. Furthermore, increasing the freedom of choice for reviewers might increase their effectiveness.

F. Extrinsic Motivation

Perhaps the most interesting are the extrinsic motivators that affect the code review, as they are the most controllable mechanism in place. We analyze them in more detail than the previous findings. Table IV summarizes the six extrinsic motivators identified in our data. We devote a paragraph to each below. We close each of them with a hypothesis on how it could be exploited by project managers implementing code review.

Reciprocity. In contrast to our present economy system, which is based on quantifiable and measurable exchange transactions using money as unit for measurement, the gift economy is much more flexible. When giving, the contributor is owed [22]. There is an implicit moral obligation to reciprocate the gesture of giving [23], but more subtle than just give-and-take. The exchange does not require quantification and measurement, or at least not explicitly. A FOSS community gives you learning, expertise and the sense of togetherness. Your respond helping others to experience the same. *"I feel satisfied when I'm able to give back something that I have"* (Participant 6).

How shall one operationalize this motivator? First, the situation where learning, expertise, and the sense of community is offered to contributors and reviewers is easy to mirror in other projects (both closed and open source). Second, our data shows that it may be worth to make the software engineering environment a relational (where relations grow through reciprocal exchanges), not only transactional (where time and effort is exchanged for a paycheck).

Participation in success. Success is attractive to hackers. A successful project earns the respect of contributors, and attracts more contributors. *"It's a great piece of software and a success story. Everybody wants to be part of it"* (Participant 16). Success is a general motivator, also for parties during code review, which is seen as a key practice contributing to the success in FOSS communities. It is hard to make a project successful, however successful projects can exploit it to attract community members, and can expect members to be more motivated to contribute.

Status in the community. Reputation. Reputation is the most pronounced extrinsic motivator in our data, out of all listed in Table IV. This is in line with the rich literature about desire for reputation as a motivation to participate in FOSS communities overall [14], [24]–[26]. Some subjects partake in code review as a way to gain reputation and recognition. Some go as far as to name the status an *"award"* for code review. Some perform it particularly diligently, as not to loose the hard earned reputation. Work experience in an open source project accumulates technical expertise and social capital elevating the contributor's status. Since the management hierarchy is much less important than in commercial organizations, the status matters more here.

Reputation is motivating even for junior contributors. Participant 1 reports how he felt having received praise from his mentor for suggesting an alternative architectural solution in a code review. The mentor recognized his technical expertise: *"I felt like I [was] made for this, I had to do this*

more and more ... four to five hours continuously, I coded for the second issue to get the same feedback and I'm still doing this continuously because that energizes me."

Paradoxically, code reviewing earns reputation, but then it may diminish the effect of code review on one's contributions. Some highly recognized senior contributors admit that they are treated more respectfully in code review, because of their reputation, or even that they are able to commit code without review.

How to exploit reputation and status in project organization to boost code review? One simple idea is to use gamification. Another idea is to let the code review practice contribute to building a meritocratic structure in the project.

Today reputation is often gamified through some kind of points system (karma), computed by the project development platform. ROS community members, who work on implementing the new code review process, believe that performing code reviews should be reflected by rewards in such a point system.

Accruing and recognizing status based on technical contributions leads to construction of meritocracy as a backbone social hierarchy in the project. The senior members of ROS community believe that meritocracy should be designed and blended into the process of code review. So code review should become an instrument into organizing the community—then, as a side effect, the code review will work better itself.

Career building. Subjects recognize that active participation in FOSS projects is a differentiating factor on the job market, a tangible sign of expertise that can be leveraged in career development. *"It's nice to build up a CV, it's nice for an intellectual perspective because once you get some code accepted, it means that you are reaching some level of, you know some stuff"* (Participant 17). A company director active in the ROS community meetings stated that his company benchmarks candidates using community profiles and reputation.

Learning. Participants of code reviews learn from each other. Subjects agree that is beneficial for many; for junior project members, but also for the reviewers. *"that's how they [reviewers] learn, maybe they will see in the code something that they didn't know about and it's interesting to them so they will ask about it, or they discuss it"* (Participant 4). *"Review is making us better programmers. We learn when we review others code and when our code is being reviewed. You learn from other code and how they code and you learn from the feedback"* (Participant 14). This is the self-applying motivator of code review: people participate for the direct educational benefit of it. Project managers should remember that for many subjects not the mundane and simple, but the stimulating and developing tasks are motivating, and maintain the corresponding allocation of tasks.

Be amongst the best. Mediocrity is prosecuted with a harsh language and strong tone in the Linux Kernel community. This attitude aims at filtering the best and shields the project from average contributions. *"I think if you get just the best people, perhaps the contributions are then the best. There are many ways of interacting that requires this high touch."* (Participant 16). While widely criticized online,[6] even the critiques admit that code review communication needs to be harsh. The problem they criticize is not harshness, but lack of respect: *"I need communication that is technically brutal but personally respectful"* (*idem.*). This is the only negative motivator observed in the study; the main form of punishment for under-performing used in FOSS communities.

[6]For example: https://sage.thesharps.us/2015/10/05/closing-a-door/

> ***Observation 6:*** *An established reciprocal gift culture, sharing in the fame of success, reputation, public visibility of status for employers, learning opportunities, and punishment for not performing ultimately the best are the key extrinsic motivators behind work and code review of our subjects. These are all non-monetary motivators that can be used to improve code review.*

G. Trustworthiness of the findings

The validity of qualitative research is achieved through trustworthiness [27], [28]. Four constructs are used to establish trustworthiness: credibility, transferability, dependability, and confirmability.

Credibility establishes internal validity, which rivals hypotheses exclusion [27], [29], [30]. It ensures the proposed theory is reliable and representative of the raw data [31]. We used peer debriefs and participant checks. One author conducted the coding the other authors confirmed the emerging theory and categories from the collected data. Participant checks have been used for narrative accuracy and interpretive validity [27], [29], [32]. Participants were asked to validate the authenticity of the verbatim transcripts. They were also asked to comment on the analytical interpretation. Their comments served as a check on the viability of the coding.

Transferability refers to the extent to which the findings of qualitative research, either partially or completely, can be generalized or applied to similar settings [27]. We believe that we meet the transferability requirements by providing evidence that the research findings could be applicable to other similar contexts (i.e., free and open source communities). An audit trail is available and detailed enough to allow other researchers to replicate a similar inquiry in similar communities [33]. Sikolia et al. [28] suggest that researchers can ensure transferability by describing the research clearly, explaining the diverse experiences of the participants, implementing methodology, interpreting the results, and adding contributions from debriefing.

Dependability is synonymous with reliability in the traditional quantitative research. It is concerned with the ability of the research to reach the same conclusions if replicated in the same setting and conditions [27]. It measures replicability or repeatability. This is done by a peer researcher who audits and confirms that the research procedures are followed and authentic. Shenton [27] suggests that the research report should include discussions on dependability and that researchers should comprehensively explain the research design and the data gathering methods.

Confirmability refers to real objectivity in the study. It is improved by triangulation of the study data and findings [27]. The study should reflect the preferences of the participants and

not the researchers. Unlike quantitative studies, the direction of a qualitative study is created by the participants and not the researchers. The reflective discussion of the researcher promotes the reality that the data indeed reflects the participants' views and not the researcher's. An audit trail should also be discussed in the study report as another tool to improve confirmability in the study [27], [29]. This audit establishes confirmability [28]. An audit trail is when a detailed process of data collection, data analysis, and interpretation of the data has been provided.

V. Discussion

Our analysis shows that the human aspect of code review is definitely not to be ignored. Human constructs such as handling rejection, coping with close scrutiny, ethic of passion, ethic of care, intrinsic and extrinsic motivations shape the execution of code review in important and mostly positive ways. We aggregate the most actionable consequences of the study, along with proposals of actionable interventions in Table V.

Rejections are an inherent aspect of the code review process (Observation 1). Software engineering environments should institute a culture where rejections are embraced. A rejection culture implies understanding and communicating that rejection is not a failure. Otherwise it is very difficult to use code-review to improve quality. Instead of being eliminated, rejection and the negative experiences need to be sublimated into a learning opportunity. Fortunately, this can be rationalized and trained and many organizations use internal reviewing successfully as way to raise quality of products.

According to Burke and Fiksenbaum passion enhances mental and psychological well-being of the employees [34]. FOSS contributors (Observation 3) show as the most passionate workers in engineering, and, as such, an excellent subject to study this phenomenon further. Project and community managers should definitely not ignore but cherish and support this virtue of software teams.

While the ethic of care is not really associated with a stereotypical antisocial programmer in public perception, the FOSS contributors clearly exhibit traits of caring, at least in the limited scope of their project and community (Observation 4). That care is apparently developed through existing mechanisms in the FOSS communities that can also be used by others: the contagious care of project leaders, mentoring arrangements, and a sense of shared ownership of project's design, goals, and ways of working.

It is difficult to directly implement exploitation of intrinsic motivators in code review. Self-determination theory attempts to differentiate factors that facilitate and that undermine intrinsic motivation [35]. A sub-theory, the cognitive evaluation theory, maintains that interpersonal events that lead to feelings of competence enhance intrinsic motivation when they are accompanied by a sense of autonomy [35], [36]. Autonomy is described as the leeway that is given to the employee to complete their job tasks [37].

Scientists do agree that satisfaction and performance increases intrinsic motivation. Kraiger, et al. [38] argue that a positive effect increases people's enjoyment and interest. Erez et, al. [39] found that a positive effect increases the intrinsic attractiveness (i.e. goodness) of moderately desirable rewards. It also affects satisfaction and performance during the activity [39], [40]. Thus it would be extremely valuable to explore more action oriented research involving intrinsic motivation in code review, and collaborative software development in general.

The richness of extrinsic motivators is visible in the studied communities (Observation 6). There are known results that extrinsic motivators do correlate with performance and that they synergize with intrinsic motivations [41], [42]. We list some ideas on how to exploit them to improve code review in the bottom most part of Table V.

VI. Related Work

The subject of code review has been investigated from various angles, yet, the human and social aspects of the process received little attention. Bachelli and Bird [43] found that the top motivation for code reviewers is finding defects, but in fact, defect-related communication is proportionally small. Instead, the reviewers are concerned with knowledge transfer, increasing team awareness, and creating alternative solutions. The fabric of code review is communication, knowledge transfer, praise and critique. Code review is primarily a social activity.

Lussier [44] describes the experience of first rejection of a contribution to the Wine project, an open source implementation of the Windows API. While the team was initially resentful, after three more rejections, the code was accepted. Meanwhile the team developed a real sense of ownership and pride in their work. Lussier recounts that the passion of contributors is one of the reasons for the high quality of code. These findings are in accordance with our conclusions.

Asynchronous reviews support team discussions and find the same number of defects as collocated meetings [1]. They provide passive listeners with learning experience; focus better on the ideal solution, not on the defects, but still find defects earlier than the scheduled reviews. Most FOSS reviews begin within hours of submitting the change and are completed within one or two days. It is key that the reviewed changes are small, independent, and complete; typically 11—32 lines of code. Rigby et al. point out that communities allow expert developers to self-select submissions to review [1]. Selecting tasks can allow experts to stay vested in a project. However, their study misses the human and social factors of code review. We show that these are important. They sway the execution positively and steer the outcome to higher quality. They also appear to be exploitable in project organization.

Votta [45] suggests that face-to-face code review meetings of the whole team should be replaced with depositions. A deposition is a three-person team: an author, a moderator, and a reviewer. To save costs, the moderator can be eliminated. The results of such meetings are just as effective as full-team meetings. Our study indicates that human and social aspects of code review can counteract the impersonality of asynchronous meetings, and they can still allow for many experts to interact.

German and colleagues [46] studied OpenStack. They found twenty-four percent of participants subject to reviews stated that

TABLE V
AN INTERPRETATION OF THE STUDY: SUGGESTIONS OF ACTIONABLE INTERVENTIONS

Consequence of Observations	Examples of consequent actions/interventions for projects
A working environment valuing code review *embraces rejection*, as rejection is inherent and valuable in code review. (cf. Observation 1)	– Provide code review guidelines on how to communicate constructive feedback and how to interpret feedback. – Include handling rejections into training for new engineers. Rejection is not failure. – Establish an in-house counseling, mentoring or coaching function for engineers that maintains understanding of the positive value of code review.
Iterative improvement is an important method of programming work, besides the pervasive striving for perfection. (cf. Observation 2)	– Promote code review practice as a knowledge sharing and learning opportunities. – Reward and appreciate learning, and striving for excellence both in teams and individuals. – Democratize the code review process. Allow reviewers to select what they want to review.
The *ethic of passion* should be nurtured in project teams and communities. (cf. Observation 3)	– Recruit passionate developers if possible. Seek passion for code review in particular. Appoint passionate individuals to lead software engineers. – Eliminate toxic sources that deplete passion from engineers. – Reduce tasks that do not coincide with passions. – Include passion and self-determination as a parameter when allocating tasks. FOSS contributors choose what to work on, what to review, when to do it, etc.
The *ethic of care* is a positive contributor in a project team involved in code review, worthwhile cultivating. (cf. Observation 4)	– Care and commitment shall be demonstrated at every level of the software team. Leaders should review code diligently. Many FOSS project leaders are top code reviewers. – The FOSS we studied, nurture care by establishing mentor–contributor relationships that last up to five years. Developing relationships seems key to the ethic of care. – Develop a team as a community, especially regarding reviews. Develop *"our way"* of doing things and enforce it in reviews; emphasize common ownership, methods, designs and successes.
Nurturing engineers' *intrinsic motives*, altruism and enjoyment. (cf. Observation 5)	– Intrinsic motives are very difficult to exploit, and little known results of action research regarding that exist in software engineering. To the best of our knowledge, so far this is mostly an area for future research.
Many non-monetary exploitable *extrinsic motives* drive FOSS contributors to make code-review effective. (cf. Observation 6)	– Do not neglect non-monetary non-material rewards, but develop a relational environment in your team, where rewards come from collaboration. – If your project is successful, use it to attract motivated community members. – Build a meritocratic hierarchy, a key motivator for performing well in code reviews. – You might want to use a point system to gamify gaining reputation and status. – Acknowledge and reward individuals and group achievements regularly, with growth opportunities (promotions, conference talks, etc.). – Use code review as a key learning and development opportunity for engineers in your team.

they are treated unfairly occasionally and fifteen percent feel they are treated unfairly often. Reviewers who were questioned stated that they conduct reviews fairly (60%), but some stated that they conduct reviews unfairly occasionally (40%). They stated that contributions are prioritized for review by the developer's expertise, the importance of the patch, the author of the patch, the difficulty of the patch, or the freshness of the patch, which can affect consistency and perceptions of fairness. Newcomers are often treated with a negative bias, as well. These findings emphasize the importance of the human and social aspect of code review.

VII. CONCLUSION

We had set out to explore why code review works for open source software communities. We interviewed 21 open source developers from four different successful communities and collected data from meetings of a community debating introduction of code review. Having analyzed the data, we find that, besides the well known project management and QA reasons for success of the code review practice, a number of human and social aspects are key—they create a psychological and social environment that is friendly for development of successful code review interactions.

In order to encourage future work, we examined the data, and extracted patterns of management, behavior, and other elements of the FOSS work environment pertaining to code review that appear to be replaceable in other contexts (Table V). We hope that these can be a useful inspiration for project and community managers. Fore-mostly, we hope that they can inspire action research interventions into both closed and open source projects, and that this way we can obtain a better understanding how to proactively control the quality of code review.

Acknowledgment: Supported by the ROSIN project under the European Union's Horizon 2020 research and innovation programme, grant agreement No 732287. We would like to thank the interviewees for their participation and making this research possible.

REFERENCES

[1] P. Rigby, B. Cleary, F. Painchaud, M.-A. Storey, and D. German, "Contemporary peer review in action: Lessons from open source development," *IEEE software*, vol. 29, no. 6, pp. 56–61, 2012.

[2] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 541–550.

[3] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 202–211.

[4] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years," *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133–154, 2002.

[5] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Foundations of empirical software engineering: the legacy of Victor R. Basili*, vol. 426, no. 37, 2005.

[6] R. J. Vallerand, C. Blanchard, G. A. Mageau, R. Koestner, C. Ratelle, M. Léonard, M. Gagné, and J. Marsolais, "Les passions de l'ame: on obsessive and harmonious passion." *Journal of personality and social psychology*, vol. 85, no. 4, p. 756, 2003.

[7] N. Noddings, *Caring: A relational approach to ethics and moral education*. Univ of California Press, 2013.

[8] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.

[9] C. Robson and K. McCartan, *Real world research*. John Wiley & Sons, 2016.

[10] M. B. Miles, A. M. Huberman, and J. Saldana, "Qualitative data analysis: A method sourcebook," *CA, US: Sage Publications*, 2014.

[11] P. Himanen, *The hacker ethic*. Random House, 2010.

[12] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Free/libre and open source software: Survey and study," 2002.

[13] R. A. Ghosh, "Understanding free software developers: Findings from the FLOSS study," *Perspectives on free and open source software*, pp. 23–46, 2005.

[14] K. R. Lakhani, R. G. Wolf, and Others, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," *Perspectives on free and open source software*, vol. 1, pp. 3–22, 2005.

[15] R. J. Vallerand, S.-J. Salvy, G. A. Mageau, A. J. Elliot, P. L. Denis, F. M. E. Grouzet, and C. Blanchard, "On the role of passion in performance," *Journal of personality*, vol. 75, no. 3, pp. 505–534, 2007.

[16] A. Bonneville-Roussy, G. L. Lavigne, and R. J. Vallerand, "When passion leads to excellence: The case of musicians," *Psychology of Music*, vol. 39, no. 1, pp. 123–138, 2011.

[17] A. L. Toombs, S. Bardzell, and J. Bardzell, "The proper care and feeding of hackerspaces: Care ethics and cultures of making," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM, 2015, pp. 629–638.

[18] K. R. Lakhani and E. Von Hippel, "How open source software works: " free" user-to-user assistance," *Research policy*, vol. 32, no. 6, pp. 923–943, 2003.

[19] B. Luthiger and C. Jungwirth, "The Chase for OSS Quality: The Meaning of Member Roles, Motivations, and Business Models," in *Emerging Free and Open Source Software Practices*. IGI Global, 2007, pp. 147–168.

[20] R. Q. Danish, M. K. Khan, A. U. Shahid, I. Raza, and A. A. Humayon, "Effect of intrinsic rewards on task performance of employees: Mediating role of motivation." *International Journal of Organizational Leadership*, vol. 4, no. 1, 2015.

[21] J. Rogstadius, V. Kostakos, A. Kittur, B. Smus, J. Laredo, and M. Vukovic, "An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets." *ICWSM*, vol. 11, pp. 17–21, 2011.

[22] E. S. Raymond, "The Cathedral and the Bazaar," 1998.

[23] D. Zeitlyn, "Gift economies in the development of open source software: anthropological reflections," *Research policy*, vol. 32, no. 7, pp. 1287–1291, 2003.

[24] J. Hahn, J. Y. Moon, and C. Zhang, "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties," *Information Systems Research*, vol. 19, no. 3, pp. 369–391, 2008.

[25] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[26] J. Lerner and J. Tirole, "Some simple economics of open source," *The journal of industrial economics*, vol. 50, no. 2, pp. 197–234, 2002.

[27] A. K. Shenton, "Strategies for ensuring trustworthiness in qualitative research projects," *Education for information*, vol. 22, no. 2, pp. 63–75, 2004.

[28] D. Sikolia, D. Biros, M. Mason, and M. Weiser, "Trustworthiness of grounded theory methodology research in information systems," 2013.

[29] S. C. Brown, R. A. Stevens, P. F. Troiano, and M. K. Schneider, "Exploring complex phenomena: Grounded theory in student affairs research," *Journal of college student development*, vol. 43, no. 2, pp. 173–183, 2002.

[30] G. Rolfe, "Validity, trustworthiness and rigour: quality and the idea of qualitative research," *Journal of advanced nursing*, vol. 53, no. 3, pp. 304–310, 2006.

[31] D. Straub, M.-C. Boudreau, and D. Gefen, "Validation guidelines for IS positivist research," *The Communications of the Association for Information Systems*, vol. 13, no. 1, p. 63, 2004.

[32] M. Carcary, "The Research Audit Trial: Enhancing Trustworthiness in Qualitative Inquiry." *Electronic Journal of Business Research Methods*, vol. 7, no. 1, 2009.

[33] A. Cooney, "Rigour and grounded theory," *Nurse researcher*, vol. 18, no. 4, pp. 17–22, 2011.

[34] R. J. Burke and L. Fiksenbaum, "Work motivations, satisfactions, and health among managers: Passion versus addiction," *Cross-Cultural Research*, vol. 43, no. 4, pp. 349–365, 2009.

[35] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being." *American psychologist*, vol. 55, no. 1, p. 68, 2000.

[36] F. P. Morgeson, K. Delaney-Klinger, and M. A. Hemingway, "The importance of job autonomy, cognitive ability, and job-related skill for predicting role breadth and job performance." *Journal of applied psychology*, vol. 90, no. 2, p. 399, 2005.

[37] C. J. Fornaciari and K. L. Dean, "Experiencing organizational work design: Beyond hackman and oldham," *Journal of Management Education*, vol. 29, no. 4, pp. 631–653, 2005.

[38] K. Kraiger, R. S. Billings, and A. M. Isen, "The influence of positive affective states on task perceptions and satisfaction," *Organizational Behavior and Human Decision Processes*, vol. 44, no. 1, pp. 12–25, 1989.

[39] A. Erez and A. M. Isen, "The influence of positive affect on the components of expectancy motivation." *Journal of Applied psychology*, vol. 87, no. 6, p. 1055, 2002.

[40] B. M. Staw and S. G. Barsade, "Affect and managerial performance: A test of the sadder-but-wiser vs. happier-and-smarter hypotheses," *Administrative Science Quarterly*, pp. 304–331, 1993.

[41] J. Boiché, P. G. Sarrazin, F. M. Grouzet, L. G. Pelletier, and J. P. Chanal, "Students' motivational profiles and achievement outcomes in physical education: A self-determination perspective." *Journal of Educational Psychology*, vol. 100, no. 3, p. 688, 2008.

[42] L. G. Pelletier and P. Sarrazin, "Measurement issues in self-determination theory and sport." 2007.

[43] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 712–721.

[44] S. Lussier, "New tricks: How open source changed the way my team works," *IEEE software*, vol. 21, no. 1, pp. 68–72, 2004.

[45] L. Votta, "Does the Modern Code Inspection Have Value," in *Presentation at the NRC Seminar on Measuring Success: Empirical Studies of Software Engineering*, 1999.

[46] D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, "Was my contribution fairly reviewed?: a framework to study the perception of fairness in modern code reviews," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 523–534.

C

Appendix C: Paper C

Affiliated Participation in Open Source Communities

Adam Alami
Computer Science
IT University of Copenhagen

Andrzej Wąsowski
Computer Science
IT University of Copenhagen

Abstract*—*Background: **The adoption of Free/Libre and Open Source Software (FOSS) by institutions is significantly increasing, and so is the affiliated participation (the participation of industry engineers in open source communities as part of their jobs).** ***Aims:*** **This study is an investigation into affiliated participation in FOSS communities. So far, little is known about the affiliated participation and the forces that influence it, even though the FOSS innovation model is increasingly becoming a serious contender for the private investment model in many sectors.** ***Method:*** **We present a qualitative inquiry into affiliated participation in the Robot Operating System (ROS) and Linux Kernel communities, using twenty-one in-depth interviews and participatory observation data from twenty-nine community events.** ***Results:*** **Our results show that affiliated participation in these communities is constrained by several barriers: objections of senior management, protection of the company's image, protection of intellectual property, undefined processes and policies, the high cost of participation, and unfamiliarity with the FOSS system.** ***Conclusions:*** **These barriers should be addressed in any organization considering using FOSS as a significant acquisition, distribution, and development strategy.**

***Index Terms*—FOSS, Open Source Software Adoption, Open Source Software Participation, Affiliated Participation**

I. Introduction

Free/Libre Open Source Software (FOSS) was born as an informal and niché movement. By mid 90's, it became a recognized software development and distribution model. The emergence of the Internet removed the physical barriers in collaboration and accelerated FOSS growth. The traditional collocated development process ceased to be the only option—the collaboration of highly recognized experts facilitates faster progress and innovation [1]. In 2008, Hauge [2] reported that already half of the surveyed companies used open source components. The FOSS process is considered a phenomenon of the *collective action* innovation model. The increasing interest in adopting FOSS could mean that FOSS becomes a contender for the *private investment* innovation model. If this assumption is valid then it is timely to investigate commercial adoption of, and the participation in, the FOSS movement.

The value of the FOSS market is estimated by some to exceed 1.9 billion [3]. We can safely assume that FOSS code is used in many of our everyday technical gadgets, software and tools. Companies and government institutions not only use FOSS, but increasingly choose to open source the code of their products. In 2016, Walmart open sourced a version of its cloud management system. In 2011, ExxonMobil released an open source "Standards DevKit" (a developer toolkit). They wanted to foster collaboration amongst oil and gas companies. In 2016, several financial companies (Morgan, Wells Fargo, and the London Stock Exchange) launched "Hyperledger"—an open source project aiming to build blockchain-based capability to track the exchange of financial assets, including stocks, and bonds. The names of the companies involved mark an interesting shift in the attitude towards open source. This trend is observed even in government policies. Recently, in 2016, the US government released a federal code source policy. It institutes a pilot program requiring that government agencies release 20% of new custom-developed code as open source.[1]

The commercial interest mixed with the collective action work model and communitarian ideology raises gripping research questions. How does a company adopt community maintained source code? How is the engagement with the community shaped? What are the forms of participation? What makes the engagement successful and well functioning? We set out to study how individuals working for commercial companies participate in FOSS communities, the so called *affiliated participation* phenomenon:

RQ1: *What are the participation models used by companies and institutions to engage with the community?*
RQ2: *What are the barriers for employees of companies to actively contribute to FOSS as part of their main job?*

We investigate these questions using qualitative research methods, collecting data during semi-structured interviews with 21 participants and through participatory observation in 29 events and meetings. We work with two large FOSS communities: ROS and The Linux Kernel. The Robot Operating System (ROS) is a framework that is widely used in robotics. The Linux kernel is an open-source Unix-like computer operating system kernel. Both communities enjoy lively participation from many commercial actors, both contributing and benefiting from the development. We find that affiliated participation is constrained by a few barriers: senior management objection, company image, intellectual property protection concerns, undefined processes and policies, the high cost of participation, and unfamiliarity with the system. When these barriers are unmanaged and the company has a business model and strategies (i.e. product, branding) misaligned to the community processes and system of values, a passive behavior toward contributing is observed.

Ideally, when a company starts to use open source code, it should envision a community participation process and comple-

978-1-7281-2968-6/19/$31.00 ©2019 IEEE

[1] https://sourcecode.cio.gov, seen September 2018

ment it with a set of participation policies. In addition, the company software acquisition strategy should reflect that decision. Otherwise, the participation becomes passive, and the company becomes a consumer of the community produced goods without contributing back to the community. A passive participation strains the community's sustainability, it leads the community into regression which hinders growth and ability to innovate.

The paper proceeds by discussing the prior research in Sect. II. Section III presents the studied FOSS communities and argues why they are a suitable choice for understanding affiliated participation. In Sect. IV, we define our mixed research method and discuss the rationale behind it. Section V presents the key findings, and Sect. VII interprets the findings as possible actions for companies interested in improving their participation in FOSS. We conclude in Sect. IX.

II. Related Work

FOSS adoption by commercial entities: Several works show the extent of the FOSS adoption in industry, the demography of the participating companies, and the participation behavior [4]. Already in 2006, companies contributed to 97 out of the 300 most active SourceForge projects [5]. Yet the market of FOSS is difficult to size [2]. The existing attempts [6]–[10] focus on few projects like the LAMP stack itself, Linux, or end-user applications (mail or office tools). Studies from Finland, UK, Australia, and US report low FOSS adoption in the public sector—Linux, used by more than 50% of respondents, is a notable exception [11], [12]. Together with the other elements of the LAMP stack, Linux was also frequently used in other sectors 15 years ago [9]. However, the adoption varies widely across countries, sectors, and company sizes, from as low as 17.7% in Sweden and as high as 43.7% in Germany [9]. A survey on Australia's top companies reports that 26% used a varied spectrum of FOSS products in 2005 [7]. With the exception of Linux, Apache HTTP Server, and perhaps a few others, most surveys report that less than 30% respondents adopted FOSS. Less is known about the extent of the internal adoption of FOSS in these companies and the participation behavior in the community. Some of our subjects claim that a pure passive adoption is a sub-optimal form of participation, where not all benefits are realized.

FOSS-related business models: Two main business models for FOSS are (i) the *support seller*, where a company sells services associated with a FOSS project, and (ii) the *loss-leader*, where a company uses FOSS to grow the user base of an industrial software product by promoting it towards a FOSS community, typically using a free license for a variant [13]. Fitzgerald [14] identified four adoption models: *value-adding service enabler* (similar to support seller), *market creation*, *leveraging community development*, and *leveraging the FOSS brand*. How do companies actually implement these business models varies, with significant heterogeneity, especially regarding the degree of openness to FOSS [15]. This heterogeneity is reflected degree of adoption, re-use, and integration [14], [16], [17]. None of these works investigates and explains how the companies actually engage with the community; what makes their business models successful, and how the engagement with the community fuels or mitigates the risks of adopting FOSS.

Affiliated participation and community relations: The detailed qualitative aspects of commercial participation in FOSS projects has attracted relatively little attention of researchers. In a systematic literature review on commercial use of open source software [18], Höst and Oručević-Alagić list only minimal work regarding the ways companies get engaged in FOSS communities: contributions happen either through individual developers [13], [19], or by a substantial commitment.

Henkel [20] observes that for-profit organizations protect their contributions to the community selectively. They perceive active participation as overly open, unsuitable for a company. Many affiliated participants contribute out of personal interest, rather than as representatives of a company. Yet, half of the supervisors are aware that their engineers share code. Furthermore, only 22.8% of respondents describe their firm's policy towards contributing actively as encouraging participation in FOSS, and 16.8% think that it is restrictive. Even though, this study sheds some light on affiliated participation, it does not attempt to understand the participation behavior. Our research objective is to analyze this issue in depth.

Lundell et al. [21], as well as Dahlander and Magnusson [22] identified three types of relationship between companies and FOSS communities: parasitic (in which the commercial interest is indifferent to its effect on FOSS), symbiotic (mutually beneficial relationships, in which both the firm and the community gain advantage), and commensalistic (relationships between the two entities where one party, the firm, benefits from the other without affecting negatively the FOSS community). Lundell et al. [21] suggest that most relationships are symbiotic. Our study shows that the relationships differ a lot between the two studied communities—not all communities have been able to successfully develop a vibrant symbiotic environment.

Open innovation in software engineering: The FOSS movement has enabled a new kind of innovation in software intensive product development, the *open innovation*—a distributed innovation process based on managed knowledge flows across organizational boundaries. Under open innovation firms use both external and internal ideas with internal and external paths to market when working to advance their technology [23]. Activities are inbound or outbound and classified as *pecuniary* (related to competitive assets and producing rewards) vs. *non-pecuniary* (related to non-competitive assets without immediate rewards). Inbound activities use input from outside the organization and outbound activities exploit internally developed innovations [24].

According to Munir et al. [25] innovation occurs as an exchange of information about new technology, and it is one of the main drivers for collective inventions. Both cooperation and competition exist in open innovation, and this results in value creation, expanding benefits from the process, and value appropriation, as benefits are seized from the process. Value creation expands the market, and value appropriation determines the firm‘s share of the market [26].

However, FOSS is more than exchanging ideas or

TABLE I
A CENSUS OF THE PARTICIPATING COMPANIES IN THE STUDY.

Company	Company sector	No. of employees	Community	Age[Y]	Model	Revenue or Budget
1	Industrial robotics start-up	12 (FY 2018)	ROS	10	passive	€0.3 million (FY 2017)
2	Industrial robotics	102 (FY 2018)	ROS	16	passive	$1.2 million (FY 2017)
3	OSRF Foundation	29 (FY 2018)	ROS	10	active	-
4	Academic institution	5,189 (FY 2017)	ROS	177	active	€21.3 million (FY 2017)
5	Industrial robotics vendor	36 (FY 2018)	ROS	7	passive	€1.6 million (FY 2017)
6	Industrial research institute	2,602 (FY 2018)	ROS	10	latent	$583 million (FY 2018)
7	Industrial research institute	25,000 (FY 2018)	ROS	72	latent	€2.3 billion (FY 2017)
8	Industrial research institute	2,761 (FY 2018)	ROS	10	passive	$322.3 million (FY 2017)
9	Industrial robotics vendor	50 (FY 2018)	Linux Kernel	26	passive	$2.9 million (FY 2018)
10	Linux distributor	12,600 (FY 2018)	Linux Kernel	26	active	$2.9 billion (FY 2017)
11	Telecommunication	4,796 (FY 2018)	Linux Kernel	37	passive	$23.5 million (FY 2017)
12	Telecommunication	4,796 (FY 2018)	Linux Kernel	37	passive	$1.87 billion (FY 2017)
13	Software vendor	21 (FY 2018)	Linux Kernel	24	latent	$1.25 billion (FY 2017)
14	Linux distributor	1,467 (FY 2018)	Linux Kernel	27	active	$365.5 million (FY 2017)

TABLE II
INTERVIEW SUBJECTS BY COMMUNITY, COMPANIES, JOB DESCRIPTION, AND WORK EXPERIENCE.

Participant #	Company	Subject role	Country	Exp. [Y]	Nature of involvement with FOSS
ROS community members					
Participant 1	Company 1	co-founder	France	5	An organization using the core of ROS to control tailored robotics systems combining various robot components.
Participant 2	Company 2	director	USA	12	A company using ROS components to build military robotics systems
Participant 3	Company 3	core developer	USA	10	The steward of the ROS community.
Participant 4	Company 4	technical lead	Netherlands	12	A large university offering consulting and research on industrial robotics.
Participant 5	Company 5	developer	Germany	6	An organization specialized in 3D sensors that enable perception and localization for robots. Uses ROS components to develop products.
Participant 6	Company 6	developer	Singapore	8	An organization leveraging ROS components to build robotics systems for customers.
Participant 7	Company 6	developer	Singapore	10	
Participant 8	Company 7	developer	Spain	10	An organization leveraging ROS components to build robotics systems for customers.
Participant 9	Company 5	developer	Germany	10	
Participant 10	Company 8	technical lead	South Korea	13	An organization leveraging ROS components to develop robotics systems for customers.
The Linux Kernel community members					
Participant 11	Company 10	kernel engineer	Denmark	18	A Linux distributor that provides consulting and support services
Participant 12	Company 10	kernel hacker	Denmark	10	
Participant 13	Company 10	principal engineer	Brazil	23	
Participant 14	Company 10	kernel engineer	USA	10	
Participant 15	Company 10	kernel engineer	USA	12	
Participant 16	Company 11	embedded Linux engineer	Spain	5	A company packaging Linux with in-house telecommunications & hardware products
Participant 17	Company 11	embedded Linux engineer	USA	7	
Participant 18	Company 12	kernel engineer	USA	30	A developer of complex software for oil and gas industry. Bundles Linux with its products.
Participant 19	Company 13	kernel engineer	USA	10	A developer of software for the telecommunication industry.
Participant 20	Company 13	kernel engineer	USA	8	
Participant 21	Company 14	project manager	USA	30	A Linux distributor that provides consulting and support services.

information. It carries a strong personal aspect (collaboration of individuals, as opposed to collaboration of companies) and associates many risk and prejudices (for instance regarding IP protection). Our study sheds light on the internal work in this processes, and the obstacles contributors and companies face in daily engagement; many of which cannot be explained in terms of market interplay or innovation.

III. Subject Communities

We have chosen to work with two communities (ROS and Linux Kernel) that enjoy a strong industrial participation and a significant adoption in the respective industries. Both communities are accustomed to commercial participation and committed to building relations with commercial members.

The Robot Operating System (ROS) is a robotics middle-ware supporting a wide a variety of platforms that it slowly becomes a *de facto* standard in robotics. The project develops tools, libraries, component drivers, conventions, standard communication and coordination features, and implementations of essential robotics-specific functionality, for example localization or planning. A ROS-based application is composed of several ROS components complemented with application specific code.

ROS originated at the Stanford Artificial Intelligence Laboratory (SAIL). In 2007, the code was transferred to a start-up, Willow Garage, and released under an open source license. Since 2013, the Open Source Robotics Foundation (OSRF) stewards the work of the ROS community.

ROS Industrial is a branch of ROS, and a corresponding association, with focus on industrial applications. Since 2012, ROS Industrial has secured the collaboration of key players in the industry (e.g. ABB, Yaskawa, Siemens, John Deere, BMW, Bosch, etc.). For this reason, ROS is a relevant and interesting community to study the interplay and the mix of proprietary, closed source, open source, and free software development.

The Linux Kernel project develops an open-source Unix-like operating system kernel that is used across extremely many hardware platforms. Since its creation by Linus Torvalds in 1991, the project has successfully developed a sustainable community. According to the Linux Foundation, which today is the main body, stewarding the development of the kernel, the project attracted nearly 12,000 developers from more than 1,200 companies, who contributed code since tracking began in 2005. The adoption of the kernel by Android is a testimony for its commercial viability, sustainability and investment value in long term. This commercial success and the rich social environment of the kernel community leans it well to study of commercial involvement in the FOSS movement.

IV. Methods

Participation of affiliated members in open source projects is a multifaceted complex process. We approach it with interpretative deductive reasoning characteristic of qualitative methods, collecting data using in-depth interviews and participatory observations.

A. Interviews

Semi-structured interviews provide us with a reasoned interpretation of the participation process by the subjects.

Subject selection: We interviewed 21 members of the ROS and Linux communities. ROS subjects were recruited at community events in 2017 and 2018 (ROSCon, ROS-Industrial Conference, Danish ROS MeetUp). Linux subjects were approached via LinkedIn. We searched for contributors on LinkedIn, using community name and terms 'contributor'/'developer'. We contacted random entries from the search results. We asked the first four participants of the Linux community to facilitate further contacts (snowball sampling). We stopped gathering data when we reached saturation. Table I is a census of the participating companies. Linux and ROS are fundamentally important for all the involved companies. The open source code is part of their main products and services. Table II summarizes the demographics of the interviewed population. With the exception of one female (Linux) all other subjects were male. The interviewer, who also selected the subjects, had no prior relationships to the participants.

Design: Prior to conducting the interviews, we compiled an interview guide with main questions and a set of possible probing questions. Table III summarizes this structure. We commenced every interview with introduction questions, before diving into the core questions of the interview. Probing questions were evoked as needed to encourage the participant to expand a particular anecdote or add more details to the answer. We encouraged the interviewees to be unreserved and fluidly accommodated the changes in the course of discussion.

Data collection: All interviews were conducted using Google Hangouts. Face to face interviews were infeasible due to the geographical distribution of subjects (Tbl. II). Each interview lasted 40–60min and generated on average fourteen pages of verbatim. The transcriptions were approved by the subjects regarding narrative accuracy and interpretive validity.

B. Participatory Observations

The observations are part of a three years action research project in which we actively take part in improving the quality of ROS and quality assurance in ROS. The observing researcher becomes explicitly part of the process being examined [27]. Observation helps him to understand what is going on in the daily development of a particular social group. Sofaer argues [27] that it is impossible to get sufficient exposure to a group without becoming a participant—it is through interaction with the participants that a researcher can come to sense the feelings, attitudes, and perceptions of the subjects. Thus, in contrast to the interviews, which present

TABLE III
Key parts of the interview framework

intro	Can you talk to me about your community? What motivated you to participate in this community in the first place?
core	Can you discuss your company engagement in the community? How do you engage with the community? Can you discuss your company contributions and contribution process? Are there any constraints from the company or from the community to contribute?
probing	Do you have a process in place for contributing to the community? What is the management's attitude toward contributing to FOSS? What type of contributions you allowed to contribute? Can you share with us an example of your company contributing to the community? And how you went about it?

a reasoned perspective of the subjects, observations expose direct attitudes, complementing the interview data.

In the field: We embedded ourselves in the ROS community by attending community events and meetings—in total 29 sessions. We helped to establish and joined the monthly meetings of the ROS quality assurance working group. The group consist of 26 members, other than the exception of one student, all members are affiliated to robotics companies, or research institutions. We also established report with the core team. The inclusive nature of the ROS community made us feel part of it fairly quickly. We took the stance of moderate participants, which allowed us to balance between being insiders and outsiders. Table IV summarizes the participatory activities.

Data collection: We observed the community while participating. The data were collected through three techniques: (1) informal conversations, (2) direct observations and (3) participation in community events and activities. Notes were taken on-the-fly and fields notes were compiled afterwards; in total 30 field notes, each 1.5 page long on average.

C. Data Analysis

We used thematic coding. We analyzed the collected empirical material following the guidelines of Robson and McCartan [28] and of Miles et al. [29]. We examined the data line-by-line using the following questions as a lens to identify codes (open coding): 1) What is this saying? What does it represent? 2) What is happening here? 3) What is at issue here? 4) What is he trying to convey? 5) What process is being described? When answering these questions, we assigned labels to the verbatim. Table V summarizes the selected themes and how they were inferred from the data. One author conducted the coding and the other author confirmed the emerging theory and categories from the collected data. Six debriefing sessions were organized where the examination of the codes and the coding process has taken place. The outcome of the data analysis was presented to the participants for validation (i.e. member checking). We shared the whole findings with all the participants and asked for feedback. All participants, who responded to our emails, confirmed our interpretation and supported the findings.

TABLE IV
PARTICIPATORY OBSERVATION VENUES

Year	Community event	Occurrences	Size
2017	Danish ROS MeetUp	1	30 persons
2017	ROSCon 2017 conference	1	large event ($\approx$ 500)
2017	ROS Industrial Conference	1	large event ($\approx$ 200)
2018	ROS Industrial Developers Meeting	7	10 persons
2018	ROS Quality Assurance Working Group	12	23 persons
2018	Danish ROS MeetUp	1	15 persons
2018	ROSCon 2018 conference	1	large event ($\approx$ 500)
2018	ROS Industrial Conference	1	large event ($\approx$ 200)
2019	ROS Quality Assurance Working Group	4	16 persons

V. FINDINGS

A. Models of Commercial Engagement in FOSS

In response to RQ1, we identify three participation models among our subjects: passive, active, and latent.

Passive participation: We have observed several cases of *passive participation*, where *an organization leverages the community products without contributing back*. For instance, Participant 1 admits that their passive attitude was conscious and strategic: *"Our strategy was from the beginning not to contribute (...) Soon, [we] will start contributing to bug fixes... There is a sentiment among our engineers to contribute back."*

Finding the right balance between contributing and reaping benefits is difficult for FOSS adopters, who struggle to protect themselves against competition while meeting the needs of customers [25]. Clearly, some subjects opted for non-pecuniary inbound engagement. (The inbound open innovation is the exploitation of externally available knowledge [30], [31].) Huizingh reports that companies engage in inbound open innovation deliberately, mostly due to concerns with sharing knowledge [31], [32].

It is known that the perspective of the active FOSS community is different: passivity adds little value to the growth and sustainability of the community; "free riders" [33], [34], who do not contribute to the development of the community but "reap the benefits," are a concern. This probably concerns some of the engineers, who are the part of the company directly interacting with FOSS contributors in online discussions.

> ***Observation 1.*** *Some of the studied companies consciously decided to benefit from FOSS in an inbound-only manner, without contributing back. Interestingly, the engineers, who interact directly with the community, contest this decision.*

Active participation: The passive participation seems to be raised more often as an issue in the ROS community than in Linux. However, also in the ROS community some of the organizations begin to realize that passive participation is not fully productive: *"We learned our lessons! Not up-streaming is a losing strategy."*[2] To fully exploit FOSS, organizations need to find ways to benefit also from giving, for example to share the cost of maintenance, to receive community-developed fixes, compatible extensions, and new features. Companies also realize that if the profitability depends on the success of the community, the long-term health of the FOSS project is also of importance for them. Some of the companies that we interviewed in the Linux Kernel community, have successfully built an *active participation* model that depends on the community: *all their developments are up-streamed and their engineers are an integral (even core) part of the community.*

Prior research confirms that it is possible and beneficial to combine non-pecuniary and pecuniary involvement with external knowledge, sharing cost and bearing cost of innovation [35]. Open innovation provides opportunities to reduce development costs, to shorten development time, and to enrich

[2] A quote from a ROS community event; up-streaming is contributing code under an open source license, which lets the community to take over maintenance.

TABLE V
THEMES: EXAMPLES, DEFINITIONS, AND WHY THEY WERE CHOSEN

Theme	The theme indicators in our data	Example of verbatim
Objection of senior management	Reports of decisions made by upper management against contributing, for example: Participant 10 links the company's decision not to contribute to the management's limited understanding of FOSS. Hence the lack of understanding and company policies are the reasons behind the objection. We also see, that the senior management owns the objection, not the subject.	*"We usually don't contribute that much to the community... It is part of this company policies but it is hard to contribute outside the company. One side I guess is a cultural thing so let's say that our bosses they don't understand well this open source and this community ideas, they don't understand that very well."* (Participant 10) *"We had a lot of push backs from management. We've done a lot of convincing."* (Participant 7)
Company's image	Direct associations between the quality of deliverable originating in the company and the company's image, for example: Participant 7 linked the company image to the quality of contribution.	*"I guess it is also an image thing. So every time you are contributing to something that is public and you are using your company name to contribute, they [management] want to be sure that the quality or the contribution is really valuable."* (Participant 7) *"Our management is concerned about our image. There is a lot of scrutiny over contributions."* (Participant 11)
Intellectual property	IP is repeatedly discussed by subjects, often linked to management's believe that contributing reduces the competitive advantage; a side-effect of (mis)understanding the FOSS cost/benefit model.	*"The main obstacle to upstreaming our code is management concerns of loosing the competitive edge."* (Participant 17)
Undefined processes and policies	Several subjects made a direct connection to the lack of clear policies and processes being problematic (an indecision).	*"It is confusing to most people. Our policies and processes are not clear! It create confusion when you can and when cannot contribute."* (Participant 6)
High cost of participation	The cost of participation appears in both interviews and observations. The subjects are aware of the additional burden introduced by contributing to FOSS.	*"The cost of upstreaming is high. You not only have to produce good code, but good Linux Kernel code that is accepted by the community"* (Participant 18)
Unfamiliarity with the "system"	Direct and indirect suggestions of unfamiliarity with the "system": the community rules, conventions, and processes. Idiosyncratic to the Linux community.	*"It's not that simple! a successful engagement requires familiarity with the system in place. Most companies are not."* (Participant 18)

internal innovation processes [25]. Our subjects indicate that understanding of this tends to grow in organizations over time.

Observation 2. *Some subjects argue that over time it is possible, and even beneficial, to develop an active FOSS participation strategy that combines pecuniary and non-pecuniary contributions, both inbound and outbound.*

Latent participation: Some organizations exercise a compromise *latent participation* model, where the *release of internally developed features is delayed until an economic gain has been guaranteed*: *"We need to recover our internal investment first before we can open source anything."* (a ROS developer asked if his company is willing to open source newly developed features). This selectively revealing strategy relies on keeping some parts internal while releasing less profit-making assets, exploiting dual licensing and restrictive licensing.

The latent model benefits both the company and the FOSS community. Contributing parts of the development, allows to embed developers in the community, and influence its direction. Simultaneously, it allows the open source community to push the organization toward sharing more [25], as we also observe in our data. Some authors recommend selective revealing [20], contributing parts considered as a commodity while keeping the differentiating components closed. Van der Linden et al. [36] emphasize that the timing of the contribution versus the release of the feature is key—the functionality will become commodity eventually due to a constant progress of technology. Also, this strategy does create a synchronization issue between the community version of the software and the in-house instance.

Observation 3. *The latent participants neutralize risks of disclosing the differentiating IP, while still benefiting from a better embedding into community than the passive participants (a non-pecuniary inbound innovation combined with deferred non-pecuniary and pecuniary outbound collaboration).*

B. Barriers to Commercially Affiliated Participation in FOSS

Since affiliated participation occurs under the umbrella of an institution, it is performed under some constraints. While affiliated participants have to follow rules, structures, and guidelines of their employer, independent participants are free of rules and have no organizational authority to report to. In response to RQ2, we identified six barriers to affiliated participation described in the paragraphs below and summarized in Tbl. VI. Each dot on the table indicates the presence of the behavior on the corresponding community or participation model. The last three columns represent where the behavior originates from. Each dot in those columns is an indication that the institution,

TABLE VI
BARRIERS VS COMMUNITIES, PARTICIPATION MODELS, AND ORIGINS

Participation barrier	ROS	Linux	Passive	Latent	Active	Institution	Community	Individual
Senior management objection	•	•	•			•		
Company's image	•		•			•		
Intellectual property protection	•	•	•	•		•		
Undefined processes & policies	•	•	•	•		•		
High cost of participation		•		•	•		•	•
Unfamiliarity with the system		•		•	•		•	•

community or individual is the originator of the behavior. There are slight differences between what barriers are experienced by subjects in the two communities, and by subjects adhering to various participation models. Also, the barriers seem to have various sources. We return to these issues in discussions below.

Objection of senior management: Some subjects indicate that senior management shows little understanding of open source community environment, social structure and processes. They are willing to consume the community goods, but resist contributing actively. *"Our bosses don't understand well this open source and this community idea"* (Participant 8, asked why the company does not contribute to the community). Unfamiliarity with FOSS is just one of the reasons. This objection is based on various grounds. *"Our management does not support contributing back to the community. They [management] have several reasons"* (Participant 2). Active participants also admit that it takes commitment of the company, not only of the engineers, to succeed: *"My company is fully committed to the community. We upstream everything we produce internally"* (Participant 13).

The passively participating companies, are typically used to produce proprietary software and engage contractually with other parties, where risks are managed, and relationships are under control. The risks of contributing actively are unknown. It is unclear how to mitigate them and how to calculate the benefits. A passive participation is safer.

Synchronizing the product strategy and the participation model helps to realize the full benefits of FOSS participation [25], [37], [38]. Little is known about how companies need to design their business models to match different open innovation strategies. For this reason, companies may mistakenly think of open innovation as yet another "off the shelf" management practice that can be implemented almost as an add-on to existing practices and organizational arrangements in the company.

Observation 4. *Subjects in actively participating companies enjoy support of the management. Subjects in passively participating companies often indicate lack of management's support as a constraining factor; apparently caused by lack of experience and little proven business practice.*

Protection of the company's image: The FOSS contributions represent the company publicly, or at least to the respective community. The company's image is easily associated with their quality. Thus, we experience concerns that negative judgments of contributions may affect this image: *"It is also an image thing. So, every time you are contributing to something that is public and you are using the company name to contribute, they[management] want to be sure that the quality or the contribution is really valuable... there are a lot of thresholds to do that"* (Participant 8).

Businesses realize that they need to create a desirable and positive corporate image, not only through marketing resources, but also by creating positive and avoid negative situations. A passive engagement is a risk mitigation strategy that can help to shelter the company image. However, several active contributors have turned this situation around, by exploiting FOSS in branding and in attracting high quality employees. In particular, we see that once the FOSS project brand is strong (e.g. Linux), companies are more likely to try to exploit it. ROS users are much more reserved about this.

Observation 5. *Some of subjects see the FOSS community as a channel in establishing, maintaining or improving image of their brand, while others do not know how to do that.*

Protection of intellectual property (IP): We registered concerns that FOSS participation implies disclosing competitive IP. The idea of sharing source developed in-house is foreign; anything produced by an employee should remain protected in the company. Businesses are reluctant to expose the differentiating technologies and to risk loosing the competitive advantage. *"The main concern is leaking our proprietary code and any architectural design that's in the code. We use ROS but we have our own architecture on how to use ROS"* (Participant 2). *"My boss object up-streaming our work. He thinks that will reveal how we do things to our competitors"* (Participant 17).

This protective attitude (regardless if justified!) is at odds with core FOSS values: sharing and openness. In the communitarian philosophy of FOSS, withholding contributions to protect IP slows down the collective innovation process and favors a single entity. Openness is a manifestation of two cultural traits of open source communities: transparency and truth [39]. Pavlicek [39] believes that truth is a fundamental community asset [39]. He explains truth and transparency empowers the community to produce free software. This conflict of positions (community values and the protection of IP) is a ground reason for passive engagement with the community.

Henkel [20] claims that management is overly concerned about openness, concluding that a more positive attitude increases benefits of open innovation. Yet, numerous other authors advise companies to contribute with commodity features and keep differentiating factors in-house [20], [37], [40]. Bosch [41] and Van Linden et al. [36] explain that the release of commodity functionality has its advantages; companies can benefit from the reduction of the cost of maintenance and focus on the differential capabilities.

Observation 6. *Passive and latent participants object to active participation for IP loss concerns. Active participants have developed a business model that is less IP-sensitive.*

Undefined participation process and policies: Some of the studied organizations suffer from a lack of formal participation policies and governance. They think it is not important, or even not necessary, to adjust their internal processes to the community engagement. Participant 7, was asked if there is a process in place to manage contributions, replied: *"I'm talking to my management to set up a process where if we develop a driver for example we can contribute back to the community. So, I'm working on the process."*

The lack of polices and processes confuses the participating engineers. Some of them have roots in FOSS. They have contributed since they were students. Some were even hired based on the FOSS record. They still display strong hacker mentality, but are uncertain how being paid affects the engagement

with the community. Munir et al. [24] postulate that software organizations that want to benefit from open innovation via FOSS engagement need to adapt their internal processes.

> ***Observation 7.*** *Subjects, who have successfully implemented an active participation model, have aligned their internal policies to reflect the community engagement. Those with passive and latent participation have not done so.*

High cost of participation: Subjects admit that active engagement is expensive: *"The cost of getting something through this process [upstreaming] is high." "[Passive participation] is the easy way to engage with the community. It takes a lot of effort to produce code that is up-streamable"* (participants 17 and 11 resp.). Both financial and psychological costs of community engagement are high. This is most visible in the Linux case, which is known for a very high barrier of entry. The typical costs include preparing the code contributions at the expected quality, meeting coding styles and conventions, accepting rejections, and dealing with multiple review cycles, preparing documentation, and tests. Often lengthy negotiations with other community members are required.

The economic formula for the participation is not well understood. If the community engagement is not seen as a long term investment but rather as seeking "freebies," we are probably dealing with a rather short term uninformed vision. Munir et al. [24] explains that FOSS participation can be costly. Open innovation is costly and it is not always easy to start it. It should be determined by the strategic, organizational, and managerial contexts of the firm, and the benefits and costs must be evaluated. In such case it would be able to not only generate cost, but also the appropriate profit [42].

Importantly, unlike the other barriers discussed above, the cost of participation is controlled also by the FOSS community. While the large part of this cost may be inherent to the process, the community has some influence on how high the barrier of entry is, and how expensive it is to adjust to the collaboration.

> ***Observation 8.*** *The cost of participation may be high, so companies need to integrate efficiently, weighing the cost against the prospective benefits. FOSS communities should be careful not to incur undue cost, especially on newcomers.*

Unfamiliarity with the "system": We find references to the FOSS "system" in our data, and statements that the "system" is a further constraint to commercial participation. The "system" refers to the social order, rituals, norms and practices of the community. Participant 18 states: *"[to] understand the process how this works. That's a big thing."* Participant 16 concurs: *"Understanding the system is something that's take time and management doesn't see the value on that."*

The high cost of participation and the unfamiliarity with the system may be addressed by hiring engineers with prior successful engagement in the community. *"When we hire new people, we always look for a cultural fit. We look for past experience in the community. Most people stay for long time. But, there are people who do not fit culturally."* (Participant 11) The contributors should share the community values and passion for the project. *"In order to be successful, you need to have passion for the project. Whoever working for the project needs to have that passion"* (Participant 12).

> ***Observation 9.*** *Engineers inexperienced with the community culture and processes struggle to fit in and are inefficient. Hiring community members counteracts this and tightens the bond between the company and the FOSS project.*

VI. Trustworthiness

An important aspect of any qualitative endeavor is establishing trustworthiness [43]. Qualitative researchers establish that the findings of the study are credible, transferable, dependable, and confirmable. Trustworthiness is assured by the establishment of these four traits [43]. We will briefly discuss how we established these traits (see tbl. VII).

Credibility: The techniques we employed to address credibility are, namely, prolonged engagement, persistent observation, and methodological triangulation [44]. Peer debriefing has been used during the research process, one author conducted the analysis and the second author validated the emerging theory against the raw data. Six debriefing sessions were organized. We also assured credibility by member checking with the participants to test the findings and interpretations. We sent the interviews transcripts and description of the findings to the participants for validation.

Transferability: Transferability is the degree to which the results can be transferred to other contexts, sites or settings. [43]. In qualitative research, this quality of transferability refers to case-to-case transfer [45]. We provided thick descriptions of the research methods so that others can judge transferability [43].

Dependability: To ensure dependability we provided information that is s logical, traceable, and clearly documented [45]. When the research process is described completely, readers are better able to judge the dependability of the research [43]. If the process of the research can be audited, then it can ensure dependability [43].

Confirmability: Confirmability is the characteristic of the match between the researcher's interpretations and findings and the data, which requires the researcher to demonstrate how the conclusions and interpretations were made [45]. According to Guba and Lincoln [43], confirmability is established when

TABLE VII
Strategies and techniques employed in the study to meet trustworthiness requirements

Strategies used to establish trustworthiness	Credibility	Transferability	Dependability	Confirmability
Prolonged engagement	•			
Peer debriefs	•		•	
Observations	•			
Triangulation	•		•	
Participants checks	•			
Comprehensive and transparent research method		•	•	
Audit trail		•	•	•

credibility, transferability, and dependability are all achieved. In addition, we compiled an audit trail throughout the study. An audit trail is a documentation that provides readers with evidence of the decisions and choices we made, including theoretical and methodological issues in the study and a clear rationale for all decisions. The audit is useful for other researchers to follow the decision trail and reach the same conclusions.

VII. Discussion: How to Overcome the Barriers?

We now enter a more speculative mode of reasoning, and consider what actions and solutions emerge from our data that help companies and communities to overcome the barriers.

Table VI shows how the participation barriers map to models in our data. Clearly, subjects following different models were focused on different barriers. The active participants have likely been able to overcome the first four barriers that relate to company's management. A promising picture emerges that organizations might be able to evolve from passive participants, through latent contributors, to full active community members, who (as per reports of our subjects) find the participation beneficial. Barriers originate not only in the institutions, but also in the communities and individuals (Tbl VI). We stipulate that barriers need to be addressed gradually and on all sides.

In Table VIII we contrast the perspective of active and passive participants on each of the barriers. We provide examples of statements from both sides in the two middle columns and add a commentary in the rightmost column. In the following text, we summarize what actions suggest themselves.

Objection of senior management: We find that active participants have overcome senior management objections. It appears that engineers seeking active participation in FOSS projects in their work choose employers where management is committed. Software teams seeking active participation should prioritize good communication with senior management, and work towards commitment.

Company's image: While active participants leverage the community success to support their brands, passive and latent participants see the FOSS community as a liability to their image. This in itself means that the FOSS projects have an image value to be exploited—a certain quality and maturity stamp. At the same time, it is clear, that businesses interested in beneficial symbiosis with FOSS projects may want to evaluate the reputation of said projects first. A low reputation project incurs an image cost, while a high reputation project can be used in branding more easily.

Protection of intellectual property: Licensing fees are far from the only way to profit in the software industry. Hardware sales, support, consulting are known reliable streams of income. Furthermore, in fast moving software sectors, speed of innovation may be more important than any temporary technological advantage. Thus, some of the active participants foresee releasing their features' IP as a strategic trade-off.

Based on this study, we can recommend to identify a suitable business model, consistent with the FOSS participation model. If indeed stringent IP protection is key to profitability, we cannot recommend active participation. However intermediate forms, where features are released with delay, or non-critical features are contributed, can already enable benefits of FOSS participation, such as lower development cost, higher quality, and using the community for growing the market share.

Undefined participation process and policies: Participation processes and policies should be documented and communicated to the engineers, regardless of the participation model. Lack of defined participation processes and policies confuses the engineers, who need transparency regarding what can and what should be done when representing a company in the FOSS community. Moreover, lack of policy increases unnecessary risks, like premature release in the latent or passive participation models. In the active model, engineers need clear feedback that up-streaming and release engineering are indeed seen by the company as legitimate use of time—otherwise the benefits might not be fully unlocked. Finally, clear policies help the participating engineers to distinguish goals of the open source community and of the company—these do not have to be the same, and they do influence engineering decisions. This awareness should be used to affect the direction of the community.

High cost of participation: The cost of participation is primarily generated by the FOSS community itself, thus we recommend that communities consider whether some costs are not undue. For instance emotional costs (e.g. when communicating a patch rejection) can be reduced by using constructive language and avoiding hostilities. The ROS community is discussing associating mentors to newcomers, who can help them in the integration process. Some communities offer training materials, or training courses. Stewarding foundations accept donations to pay experienced members for some work, instead of using the in-house engineers who may incur a higher overhead if inexperienced.

Unfamiliarity with the "system": Similar measures should be taken as for the cost of participation. Companies in both studied communities, and at all participation models, recommend hiring experienced FOSS engineers, preferably directly from the community. This entirely elevates this barrier, and it has a side effect that it tends to bring technical excellence to the company. In addition experienced FOSS contributors could mentor inexperienced employees, who have little experience in dealing with FOSS communities.

The participation of a commercial and non-commercial institutions in a FOSS community occurs through an employee or a group of employees. Sometimes, the affiliation remains anonymous: *"You sign up with your name not with the name of the company. People know my name not the company."* (Participant 12) However, in some instances, the member is known in the community to represent a company. Participation mechanisms in the FOSS communities have been originally created for individuals not for organizations. The increase in commercial participation calls for a change to the community participation model. For example, the sign-up form for the ROS community online forum is designed to capture individual demographic data only, but neglects the fact that the member signing up may be working for a company.

TABLE VIII
CONTRAST ANALYSIS: PARTICIPATION BARRIERS MANIFESTATION AND HOW THEY ARE RESOLVED IN SUBJECT ORGANIZATIONS.

Barrier	Manifestation (Passive Perspective)	Resolution (Active Perspective)	Researcher's Commentary
Objection of senior management	*"Our bosses they don't understand well this open source and this community ideas."* (Participant 10)	*"Our management is committed to the Linux community [...] very supportive of the community... We communicate all the time with management"* (Participant 11)	Poor understanding (passive) may be due to a communication failure. Managers in the active organization established a good communication channel with engineers.
Company's Image	*"Every time we want to contribute, our management object and each time the excuse is the quality many not be good enough for our company's image."* (Participant 12)	*"We believe contributing is good for our brand. It's a strong and successful project"* (Participant 15)	The employer of Participant 6 (passive) uses the risk optics to assess the cost, while the employer of Participant 15 (active) consciously exploits the reputation of Linux to strengthen own brand.
Protection of IP	*"The main concern is to reveal our architecture to our competitors."* (Participant 16)	*"To strike an effective balance between open source and proprietary code, the key is to think strategically... We engage with a strong community to help us with the process, identify bugs, and maintain a steady pace of new feature releases. Not developing these capabilities exclusively in-house frees up our engineers to focus on projects that really drive the business."* (Participant 15)	The passive participant clearly values the protection of IP more than the cost savings and the open innovation potential. The active participant has made a strategic decision to adopt FOSS both inbound and outbound, lowering cost and accelerating innovation.
Undefined processes & policies	*"We do not have an internal process or policies in place to tell us how and when to contribute."* (Participant 10)	*"Our policy is to upstream everything."* (Participant 11)	The passive participant suffers from lack of clear policy. The employer of Participant 11 (active) has a clear policy, that leaves no doubt to the engineers.
High cost of participation	*"It's not easy to contribute and when we do we get told it's not relevant or its quality not good enough. This process is very costly for us"* (Participant 18)	*"We are an integral part of the community."* (Participant 11)	Participant 11 has integrated into the community, learned its processes and accepted costs, including the initial hurdles. See also the comment under IP above.
Unfamiliarity with the "system"	*"When we hire cultural fit is very important... It takes time to get familiar with the system"* (Participant 14)	*"We look for past community engagement and participation. It is important to us the community exposure."* (ROS QA working group member)	A cultural fit and familiarity with the system is important to both inbound and outbound FOSS adoption.

VIII. LIMITATIONS

We briefly discuss the limitations of this study. First, the findings may not translate to other communities. Although, ROS and Linux exemplify commercial participation in FOSS, the behavior described in this paper may take different form in other communities. Still note that we observed a saturation of material with last interviews and events. Second, our participatory data only covers ROS. This make the data skewed toward ROS. But was valuable for triangulation. Third, robotics and operating systems are two distinctly different domains with two recognizably different populations of participants, consumers and vendors. These products target different markets with distinct dynamics. Interview and observation data reveal participants perspective, but do not capture the market dynamics and its influence on participation. In addition, with the exception of three participants (co-founder, director and project manager), most of our participants are developers. Finally, ROS and Linux are different in their development journey. While the Linux community is 28 years old, the ROS community is hardly 12.

IX. CONCLUSION

We have identified six participation barriers and discussed how they affect the three participation models, discussing them against known research. According to the body of knowledge on open innovation, the four interlocking elements of the business model are customer value proposition, profit formula, key resources, and key processes [46]. Our data confirms that these elements need to be understood and arranged in order for the company to reach the maximum level of beneficial involvement in FOSS. The high cost of participation prevents success if the company's value proposition is not sufficiently linked to an open source project. IP protection rules out open active participation for companies whose market advantage and profit is based on technology confidence. Management needs to devote resources and regulate participation to fully exploit the collaboration with FOSS—not just download free source code, but also to share the maintenance burden, receive bug fixes and new features to the up-streamed code. Depending on whether these conditions can be met, companies settle on passive, active, and latent participation models.

While there is lots of speculation regarding FOSS-based business models in open source advocacy writing, relatively little solid and documented patterns can be found in research works. This study has identified that active and latent participants are a good source of subjects to systematically collect and document such successful business models, to the benefit of software companies considering FOSS involvement.

The robotics market has been extremely lively the last ten years, when the technology begun to meet the wider commercial applicability threshold. Most companies are small start-ups, complemented by few established machine and automation technology giants. Given the expected growth of commercial robotics, it is very interesting to investigate the business models for robotics companies adopting FOSS.

Acknowledgments: Work supported by the EU's H2020 research and innovation programme, grant No 732287 ROSIN. We thank the interviewees for making this research possible.

REFERENCES

[1] G. Anthes, "Open source software no longer optional," *Communications of the ACM*, vol. 59, no. 8, 2016.

[2] Ø. Hauge, C.-F. Sørensen, and R. Conradi, "Adoption of open source in the software industry," in *IFIP International Conference on Open Source Systems*. Springer, 2008.

[3] M. Michlmayr, F. Hunt, and D. Probert, "Quality practices and problems in free software projects," in *Open Source Systems*, 2005.

[4] J. West and S. O'Mahony, "Contrasting community building in sponsored and community founded open source projects," in *HICSS'05.*, 2005.

[5] A. Bonaccorsi and C. Rossi, "Why open source software can succeed," *Research policy*, vol. 32, no. 7, 2003.

[6] R. A. Ghosh, "Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU," 2007.

[7] S. Goode, "Something for nothing: management rejection of open source software in Australia's top firms," *Information & Management*, vol. 42, no. 5, 2005.

[8] U. Nikula and S. Jantunen, "Quantifying the interest in open source system: case south-east Finland," in *1st International Conference on Open Source Systems (Scotto, M. and Succi, G. Eds.)*, 2005.

[9] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Free/libre and open source software: Survey and study," 2002.

[10] D. A. Wheeler, "Why open source software/free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers," 2007.

[11] T. Waring and P. Maddocks, "Open Source Software implementation in the UK public sector: Evidence from the field and implications for the future," *Intl. Journal of Information Management*, vol. 25, no. 5, 2005.

[12] M. Välimäki, V. Oksanen, and J. Laine, "An empirical look at the problems of open source adoption in Finnish municipalities," in *7th International Conference on Electronic Commerce*. ACM, 2005.

[13] F. Hecker, "Setting up shop: The business of open-source software," *IEEE software*, vol. 16, no. 1, 1999.

[14] B. Fitzgerald and T. Kenny, "Developing an information systems infrastructure with open source software," *IEEE Software*, vol. 21, no. 1, 2004.

[15] A. Bonaccorsi, D. Lorenzi, M. Merito, and C. Rossi, "Business Firms' Engagement in Community Projects. Empirical Evidence and Further Developments of the Research," in *Emerging Trends in FLOSS Research and Development, 2007. FLOSS'07.* IEEE, 2007.

[16] Z. Obrenovic and D. Gasevic, "Open source software: All you do is put it together," *IEEE software*, vol. 24, no. 5, 2007.

[17] Ø. Hauge, "Adoption of Open Source Software in Software-Intensive Industry," Ph.D. dissertation, PhD thesis, 2010.

[18] M. Höst and A. Oručević-Alagić, "A systematic review of research on open source software in commercial software product development," *Information and Software Technology*, vol. 53, no. 6, 2011.

[19] K. R. Lakhani, R. G. Wolf, and Others, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," *Perspectives on free and open source software*, vol. 1, 2005.

[20] J. Henkel, "Selective revealing in open innovation processes: The case of embedded linux," *Research policy*, vol. 35, no. 7, 2006.

[21] B. Lundell, B. Lings, and E. Lindqvist, "Perceptions and uptake of open source in swedish organisations," in *IFIP International Conference on Open Source Systems*. Springer, 2006.

[22] L. Dahlander and M. G. Magnusson, "Relationships between open source software companies and communities: Observations from Nordic firms," *Research policy*, vol. 34, no. 4, 2005.

[23] H. Chesbrough *et al.*, "Open innovation," 2003.

[24] H. Munir, J. Linåker, K. Wnuk, P. Runeson, and B. Regnell, "Open innovation using open source tools: a case study at sony mobile," *Empirical Software Engineering*, vol. 23, no. 1, 2018.

[25] H. Munir, K. Wnuk, and P. Runeson, "Open innovation in software engineering: a systematic mapping study," *Empirical Software Engineering*, vol. 21, no. 2, 2016.

[26] E. van Burg, C. Giannopapa, and I. M. Reymen, "Open innovation in the european space sector: Existing practices, constraints and opportunities," *Acta Astronautica*, vol. 141, 2017.

[27] S. Sofaer, "Qualitative methods: what are they and why use them?" *Health services research*, vol. 34, no. 5 Pt 2, 1999.

[28] C. Robson and K. McCartan, *Real world research*. John Wiley & Sons, 2016.

[29] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative data analysis*. Sage, 2013.

[30] U. Lichtenthaler and E. Lichtenthaler, "A capability-based framework for open innovation: Complementing absorptive capacity," *Journal of management studies*, vol. 46, no. 8, 2009.

[31] E. K. Huizingh, "Open innovation: State of the art and future perspectives," *Technovation*, vol. 31, no. 1, 2011.

[32] D. Kline, "Sharing the corporate crown jewels," *MIT Sloan management review*, vol. 44, no. 3, 2003.

[33] W. Scacchi, "Computer game mods, modders, modding, and the mod scene," *First Monday*, vol. 15, no. 5, 2010.

[34] M. L. Markus, "The governance of free/open source software projects: monolithic, multidimensional, or configurational?" *Journal of Management & Governance*, vol. 11, no. 2, 2007.

[35] S. W. van Rooij, "Open Source software in US higher education: Reality or illusion?" *Education and Information Technologies*, vol. 12, no. 4, 2007.

[36] F. Van der Linden, B. Lundell, and P. Marttiin, "Commodification of industrial software: A case for open source," *IEEE software*, vol. 26, no. 4, 2009.

[37] K. Wnuk, D. Pfahl, D. Callele, and E.-A. Karlsson, "How can open source software development help requirements management gain the potential of open innovation: an exploratory study," in *ESEM'12*, 2012.

[38] W. Stam, "When does community participation enhance the performance of open source software companies?" *Research Policy*, vol. 38, no. 8, 2009.

[39] R. Pavlicek and R. Foreword By-Miller, *Embracing insanity: Open source software development*. Sams, 2000.

[40] J. West and S. Gallagher, "Challenges of open innovation: the paradox of firm investment in open-source software," *R&d Management*, vol. 36, no. 3, 2006.

[41] J. Bosch, "Achieving simplicity with the three-layer product model," *Computer*, vol. 46, no. 11, 2013.

[42] F. Michelino, M. Caputo, A. Cammarano, and E. Lamberti, "Inbound and outbound open innovation: organization and performances," *Journal of Technology Management & Innovation*, vol. 9, no. 3, 2014.

[43] E. G. Guba and Y. S. Lincoln, "Naturalistic inquiry (Vol. 75)," *Beverly Hills, CA: Sage*, 1985.

[44] N. K. Denzin, *Sociological methods: A sourcebook*. Routledge, 2017.

[45] S. J. Tracy, "Qualitative quality: Eight "big-tent" criteria for excellent qualitative research," *Qualitative inquiry*, vol. 16, no. 10, pp. 837–851, 2010.

[46] T. Saebi and N. J. Foss, "Business models for open innovation: Matching heterogeneous open innovation strategies with business model dimensions," *European Management Journal*, vol. 33, no. 3, 2015.

D

Appendix D: Paper D

Noname manuscript No.
(will be inserted by the editor)

A Tailored Participatory Action Research for FOSS Communities

Adam Alami · Peter Axel Nielsen · Andrzej Wąsowski

Received: date / Accepted: date

Abstract Participatory Action Research (PAR) is an established method to implement change in organizations. However, it cannot be applied in the open source (FOSS) communities, without adaptation to their particularities, especially the specific control mechanisms developed in FOSS. FOSS communities are self-managed, and rely on community consensus to reach decisions. This study aims to propose a PAR framework specifically tailored to FOSS communities. We successfully applied the framework to implement a set of quality assurance interventions in the ROS community. The framework we proposed is composed of three components, interventions design, democratization and execution. We believe that this process will be amenable to other FOSS communities. We have learned that changing a particular aspect of a FOSS community is arduous. To achieve success the change must rally the community around it for support and attract motivated volunteers to implement the interventions.

Keywords Participatory Action Research · FOSS · Change implementation

1 Introduction

Many FOSS (Free and Open Source Software) projects have matured over the years to produce software of considerable size, complexity and some have seen generational changes. The need to rediscover, refactor, and re-engineer existing code bases will thus increase over time [8], as will the need to deal with technological changes, processes, infrastructure, dependencies, and deployment platforms. We can safely assume that handling growth and maturity will require use of the best practices of software engineering methods and tools. An increase in popularity of a FOSS project implies that the community has to mature and improve its processes and practices. However, our understanding of how to change a FOSS community is limited.

The objective of this study is to improve the Robot Operating System (ROS) community quality assurance (QA) practices. Through the mechanisms of a participatory action research process, we intend to introduce, and when necessary, reinvigorate best QA practices in the ROS community. We want to align the ROS community QA with other FOSS communities' practices and with modern software engineering best practices. This will simultaneously yield a change in the affected community and new method to work with FOSS communities.

How does a FOSS community carry out change implementation? Fitzgerald [12] asserts that as communities increase maturity and size, they will adopt more formal decision making processes, such

A. Alami
Rued Langgaards Vej 7
DK-2300 Copenhagen S Denmark
E-mail: adaa@itu.dk

P. A. Nielsen
Aalborg University Selma Lagerlöfs Vej 300
9220 Aalborg Denmark
E-mail: pan@cs.aau.dk

A. Wąsowski
Rued Langgaards Vej 7
DK-2300 Copenhagen S Denmark
E-mail: wasowski@itu.dk

as voting in Apache, and establish meta-processes for creating such formal structures, and the steering board in Gnome. However, not all FOSS communities have a governance mechanism in place and even if they do, it is not understood what role these governance structures play in introducing change to the community. Our subject of study, the ROS community (see Section 5) is self-managed. Recently, the community has experienced a significant growth in its commercial participation including Amazon, Intel, Bosch, and Toyota. Many other companies are associated in the ROS-Industrial Consortium [1]. These participants have become outspoken and brought their own requirements for change. We were approached by ROS-Industrial to execute a quality assurance enhancement project in the ROS community. This paper shares the experience of two years of a four year long endeavor.

To implement this change, we needed a process and a methodological framework. We opted for the out of the box version of participatory action research (PAR). It is an appropriate choice given that action research has proven to be a reliable tool to institute change in organizations. However, we quickly realized that the PAR framework is not in tune with the ROS community values and norms and had to be adapted. The challenge is how to adapt PAR for the ROS community? We want to address the following research question:

RQ: How can participatory action research methods be adapted to carry out change in a FOSS community?

We designed PAR4FOSS a participatory action research process that caters for FOSS cultural traits. We propose to use decision-making by consensus and democratization of interventions. We designed and implemented a change process that caters for and embraces the ROS community uniqueness.

- We choose to democratize and gather a consensus around our actions via consultation and dissemination in the community.
- The change process mandates transparency. Transparency is reflected in the dissemination of decisions and solicitation of feedback from the community.
- The change process is self-managed. Although the process necessitates a facilitator, the structure is flat and relationships are collaborative.

The audience of this work is action researchers, community managers and maintainers who plan to introduce change to a FOSS community or other similar self-managed teams and organizations. We hope that they can leverage the method to introduce change to other communities and subsequently enhance and evolve the method.

We start the paper by providing a background on participatory action research and FOSS communities (Section 2). In Section 3 we introduce PAR4FOSS, a tailored participatory action research for FOSS communities. In Section 4, we highlight the research design. Section 5 is dedicated to the ROS community and a discussion of its cultural fabric. In Section 6, we describe the process that led to the design of our method. In Section 7, we evaluate the proposed method and we discuss its strengths and weaknesses. We discuss the lessons learnt in Section 8. We conclude in section 9.

2 Background

We now introduce the two main concepts driving this paper, which are Free or Open Source Software (FOSS) and Participatory action research (PAR).

2.1 Free and Open Source Software (FOSS)

Open source project refers to any software made public and open for others to modify. Bretthauer defines Open Source as "software which is licensed to guarantee free access to the programming behind the pre-compiled binary, otherwise called the source code" [6]. Open Source and source disclosure has always co-existed as a software licensing agreement, until mid-80s. The Internet removed the physical barriers and facilitated virtual collaboration accelerating software distribution. The traditional collocated software development process is no longer the only option. In mid-90s open source has become a recognizable software development and distribution model [2, 7]. The modern open source software movement began, in 1976, with Richard Stallman. He created EMACS and became a free software advocate.

[1] `https://rosindustrial.org/ric/current-members/`

In the upcoming subsections, we give an overview of what FOSS is about, and what is the social structure of FOSS communities, who are the contributors of FOSS, and what is the predominant culture of FOSS.

FOSS Structure

The structure of each FOSS community is unique, reflecting its principles, values, beliefs, and norms. The roles of members in a FOSS community have been described as an onion, with horizonal layers, rather than a vertical hierarchy. At the center are the core of developers, with the next layers consisting of the informal community managers, project managers, developers, and passive users making up the outer layer. Individuals generally move inward through merit, nominated by another member of the community, based on code contributions, facilitating others in their work, mediating conflicts, and solving problems in the community [18].

FOSS Demographics and Motivation

The scientific community investigated who FOSS participants are. Well-elaborated surveys report that the population is predominately young males, with educational or occupational background in computing, software engineering or programming. Although, the sampling of these surveys varied in sizes, the depiction of the population demographic is consistent. Around 98% of FOSS contributors are male, aged between 20 and 30 years of age [9, 14, 15].

The motivation of members to join a FOSS community is the desire for control of their work. This "free choice" of assignments is important to members of a FOSS community. The members identify with the work they produce, and it is a source of pride for them. In addition, the members of an FOSS community have free choice in work assignments. Being able to pursue one's own passions and interests is a great advantage for members of the community [11].

The underlying motivation, to participate in FOSS are either intrinsic or extrinsic [5, 16, 23]. Intrinsic motivation refers to behavior that is driven by internal rewards [29]. The participation is based on internal satisfaction and self-enjoyment [23]. Ryan and Deci [29] explain that the motivation to engage in an intrinsic behavior arises from within the individual because it is naturally satisfying. This contrasts with extrinsic motivation, which involves engaging in a behavior in order to earn external rewards [23, 30]. This type of motivation arises from outside the individual. It can involve tangible or psychological rewards. Psychological forms of extrinsic motivation can include praise and public acclaim [30].

FOSS Meritocracy

FOSS communities are meritocratic where people are valued by the quality and quantity of their contributions [13, 21]. The community members' self-image includes the traits of desiring "geek fame", a desire to build trust and reputation within the community, generosity of time, expertise, and source codes, and the desire to create reliable and quality software [31].

In a social arrangement where meritocratic filtering is a norm, people with higher merit are placed into positions of highest authority [35]. Sociologists argue that meritocratic systems encourage the most talented into the most functionally important positions [35]. Open source communities are almost entirely meritocratic. Reputation in an open source community is gained by valuable and innovative contributions to the community. A meritocracy is not merely a desire to be recognized with a higher community status. It is a complex mechanism of control and structure. Scacchi [33] defined this social community conduct as "interlinked layered meritocracies operating as a dynamically organized but loosely coupled virtual enterprise."

Our societies and communities rely on social orders, or the links found among institutions, traditions, values and morals that work cooperatively to keep societies moving forward instead of falling apart [35]. Scacchi's [33] definition resembles a social order. He explains that "layered meritocracy" is a system to control, structure and oversee the development activities. It implies certain form of authority legitimized by the gained reputation, experience and accomplishment in the community [33, 34]. There is also an implicit hierarchy where core developers are ranked higher. However, this social order is not authoritarian [32]. Consent is sought among core developers and contributors in the decision-making process [33, 34]. Leadership, in FOSS communities, is earned through experience. Core developers are usually the early contributors to the community and some of them are the founders.

2.2 Participatory Action Research

Participatory action research (PAR) is a set of consecutive actions and reflections [20]. It is a collaborative approach to research whereby the researcher partners with the subject being studied to achieve a particular outcome. The environment where the research occurs has a great influence on the course of action. Participatory research varies from conventional methodologies by focusing and acknowledging the "local" perspective and knowledge. Participatory research seeks to empower people to find solutions according to their priorities [20].

Action research becomes participatory action research when the researchers and participants are engaged collaboratively in the inquiry [20]. Participatory action research is defined by three reiterative phases: inquiry, action, and reflection [20]. The iterative nature of the process provides opportunities for improvements to the knowledge, methods, and better understanding of the subject of the inquiry. However, Kemmis and McTaggart [19] argue that the stages may amalgamate in practice as the learning experience of the inquiry progresses. PAR differs from action research by its principles of democratic participation and communal reflection. The power is distributed equally between the researchers and the participants. Participants are actively engaged in the research process. By empowering the participants of the study, PAR promotes validity. Communal reflection provides an avenue to collectively address any bias or pre-established assumptions that the researchers may have [20].

Participatory action research includes analysis of what people do, how they interact with the environment and others, what people mean in their language, what they value, and their understanding and interpretation of their world. Understanding material, social, and historical circumstances that people create and then recreate in social settings allows ideas of changes that can improve the problem. Participatory action research helps people change their practices, their understandings of these practices, and the situations in which people live and work by self-reflection. Foundations of participatory action are the following [20]:

- It is a social process that explores the realms of the individual and the social.
- It is participatory, and it engages people in examining their knowledge.
- It is practical and collaborative.
- It is emancipatory, helping people to release themselves from constraints that are irrational or unproductive.
- It is critical in that it allows people to contest and reconstitute unproductive, unjust, or unsatisfying ways of interpreting their world.
- It is reflexive in that it allows people to investigate reality in order to change it.
- It attempts to transform both theory and practice.

Participatory action research is context-specific, fluid, and inclusive [19]. Participatory action research analysis focuses attention on the central issues. The control lies, not in the researcher or the participants, but is shared, in the form of a zig-zag pathway with greater or less participation at various stages [19].

2.3 Related Work

Change of FOSS communities has, received little attention from researchers, with the exception of Krafft's [22] and Özbek's [27] PhD dissertations. While Krafft looks at influencing factors on innovation adoption in the Debian community, Özbek [27] focuses on concepts related to the introduction of innovations to FOSS communities. Although these studies are the closest to our work, both do not propose a working method, but rather demonstrate the lack of a specific method dedicated to change of FOSS communities.

Krafft followed the Delphi method. In the Debian community, innovations are built on consensus among the community. First, pioneers work on innovation, always ready to improve the tools and products. Pioneering usually begins inside the project or is carried into the project by a member who picked up the concept externally in order to address a need [22]. Krafft recommends as many as 15 factors affecting adoption of technology and tools in the Debian community. The three most relevant to our work are: (1) Sedimentation: awareness of available tools, (2) Marketing: active promotion of tools, and (3) Peercolation: acceptance by peers [22]. These factors can inform a change implementation process, but do not depict a working method to implement change.

Özbek [27] studied how innovation is introduced to FOSS communities. Proprietary software organizations use pain points and monetary incentives to encourage change, but since FOSS are voluntary organizations, finding ways to encourage change is important [27]. An innovation is a tool, method

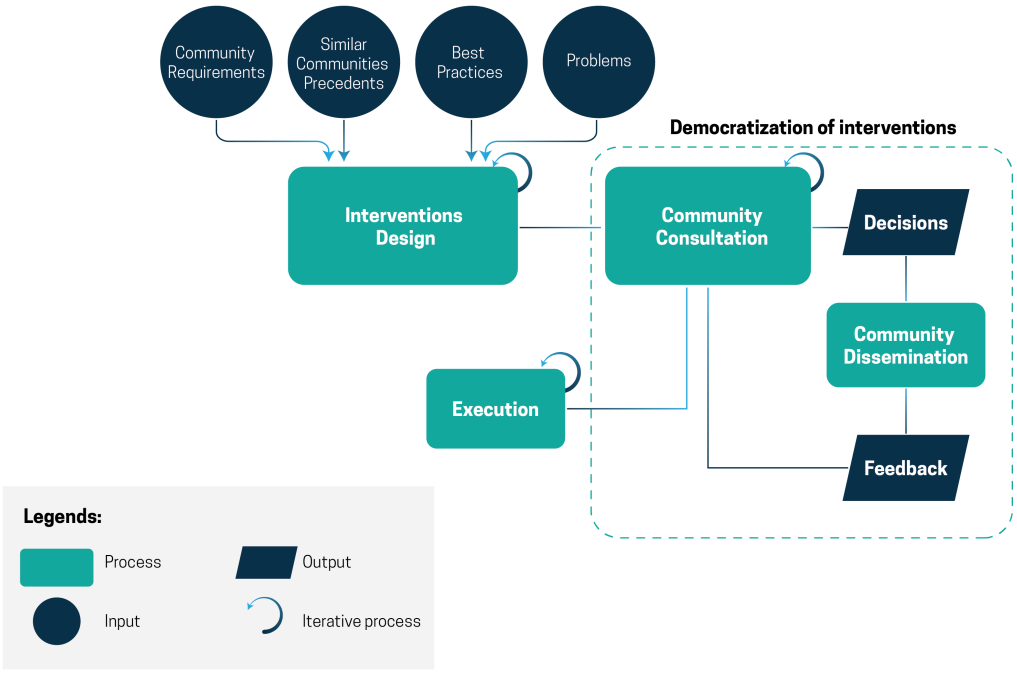

Fig. 1 The PAR4FOSS Method

or practice which can become part of a software process created by volunteers who are distributed geographically. Using grounded theory, Özbek analyzes 83 episodes of innovation in 13 FOSS projects; 37 failures, 30 successes, and 16 with an unknown outcome. Success is defined as an innovation used on a routine basis that solved a problem and attained its goal. When innovations are introduced, the result can be no adoption, no goal attainment, and failure or postponement in lieu of adoption. Partial migrations occur as well, which is when an existing innovation is replaced by a newly introduced one [27].

3 Participatory Action Research for FOSS (PAR4FOSS)

In this section, we introduce and discuss our participatory action research framework for FOSS communities.

Conscious of the FOSS communities uniqueness, we designed and executed a participatory action research process tailored for FOSS. The method (Fig. 1) we propose is composed of three components: interventions design, democratization and execution. The method (**PAR4FOSS**) is a three activities iterative process where transparency and open decisions were implemented. The structure is influenced by the self-management and collaborative aspects of the FOSS communities. We believe that this process will be amenable to other FOSS communities and will thus help introducing change in other FOSS projects.

Interventions are designed and enhanced iteratively. They are obtained from four different sources: (1) community requirements, (2) similar communities precedents, (3) best practices and (4) problems:

- Community requirements: These are legitimate needs to improve a situation or meet certain objectives.
- Similar communities precedents: Something other FOSS communities succeeded at doing and had a positive impact.
- Best practices: These are industry practices to turn into community practices in order to enhance a situation of align the community with industry practices.
- Problems: It is a specific community problem related to the scope of the change. Simply, a problem can be defined as the difference between what is happening (as is), and what should be happening (should be). The misalignment between the as-is and should-be is the “problem.”

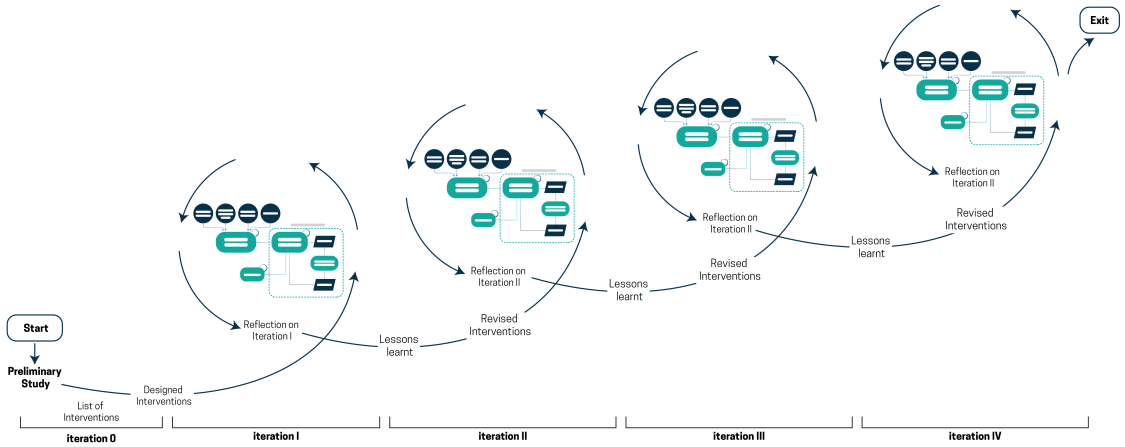

Fig. 2 PAR4FOSS Iterations

The democratization of interventions is to legitimize the interventions. As shown in Fig. 1, This legitimization is gained from community consultation and dissemination. Community consultation is a process by which the community's inputs on the interventions is sought. It aims at gathering local knowledge to test the interventions for cultural sensitivity, social effectiveness and appropriateness. The output of this activity is a set of decisions (i.e. tuned interventions and an action plan for each intervention). This consultation may occur with a selected community advocates, enthusiasts for the change and experts on the subject matter (i.e. consultative body). To ensure transparency, a wider community dissemination is recommended to gather feedback, consensus and support for the interventions. When feedback is received from the community, there is an obligation to act upon it. Feedback is communicated to the consultative body for deliberation and another iteration of interventions enhancement. At one stage, interventions will stabilize and become candidate for execution.

Execution is the actual implementation of the interventions. This process leverage existing community capabilities to implement the interventions. This is not the end point. Eventually, there will be learning from this activity that should be shared with the consultative body for knowledge acquisition and future considerations.

The PAR research is designed to be executed iteratively. Iterations consist of a set of aggregated interventions to implement change in the FOSS community. We propose a PAR process (Fig. 2) in two consecutive phases: (1) the pre-interventions phase and (2) the interventions phase were we went through four iterations [3, 4].

Pre-interventions phase: This is iteration zero in Fig. 2. We began with an empirical investigation of the research environment [10], the identification of problems and issues related to the proposed change. This is an exploratory phase which aims to discover the scope of interventions, the problem space, and the relationship between these two components. During this phase, the design of interventions should take place.

Interventions implementation phase: The interventions will be executed in iterations (four iterations in Fig. 2). Each iteration is the execution of PAR4FOSS method. Upon the completion of each iteration, we suggest conducting a reflection session to draw learned lessons and revise the list of interventions.

4 Application of PAR4FOSS in ROS community

4.1 The Project Context

ROS managed to attract a wide global community of users and contributors. ROS-Industrial is a branch of ROS with a specific industrial application focus. Incepted in 2012, ROS-Industrial Consortium has secured the collaboration of key players in the robotics industry (ABB, Yaskawa, Siemens, John Deere, BMW, Bosch, etc.) ROS-Industrial ambition is to become the worldwide open-source standard for industrial robots. ROS-Industrial stakeholders commenced raising their concerns regarding the quality assurance

practice in ROS not being aligned with FOSS communities and software engineering best practices. Although, we were originally external to the community, the motivation for the project came from within the community. Thanks to the push from the ROS-Industrial Consortium, the H2020 project ROSIN was established to enhance ROS quality assurance practices and to promote ROS as a reliable robotic platform for industrial users. This work is one of the results of this community inspired project.

Access to the research setting. During the inception of the project, we contacted the ROS core team and we entered a dialogue regarding the idea of the project. The core team assigned one of its members to the project and we recruited another contributor with over ten years experience in the community. Both have distinguished reputations in the community. We asked our informants to lead the recruitment of participants for a quality assurance working group. The aim was to leverage our informants' meritocratic status in the community for credibility. This approach worked well.

Our proposed PAR4FOSS (Fig. 1) advocates community consultation for change introduction. We contextualized this guideline in the ROS community by instituting a *community working group*. The group's mission is to collaboratively promote an extended array of quality assurance best practices in the ROS community, so that it may endeavor to carefully construct a cultural environment where quality is part of the fabric of the community. The scope of the group work is to promote and implement quality assurance initiatives, for example testing, code reviewing, continuous integration, and code scanning. The group became the problem owner and the project conduit in the community.

Defining the problem. The inquiry stage of PAR is the identification of a shared practical problem by collaborators, and methods to address the problem [19]. We conducted a qualitative study [1] during the inquiry phase, to understand the problem and the underlining causes. We present a summary of this study in Section 5.1.

Interventions. During the inquiry phase we identified the problems and proposed solutions. Each solution is an intervention. Community intervention encompasses the creation, implementation, and evaluation of best practices designed to affect the community as a whole. Table 2 lists the interventions. The list of interventions was presented to the working group for review and validation. A final list was agreed upon and published in the QA forum for a wider community consultation.

4.2 Participatory Action Research in ROS

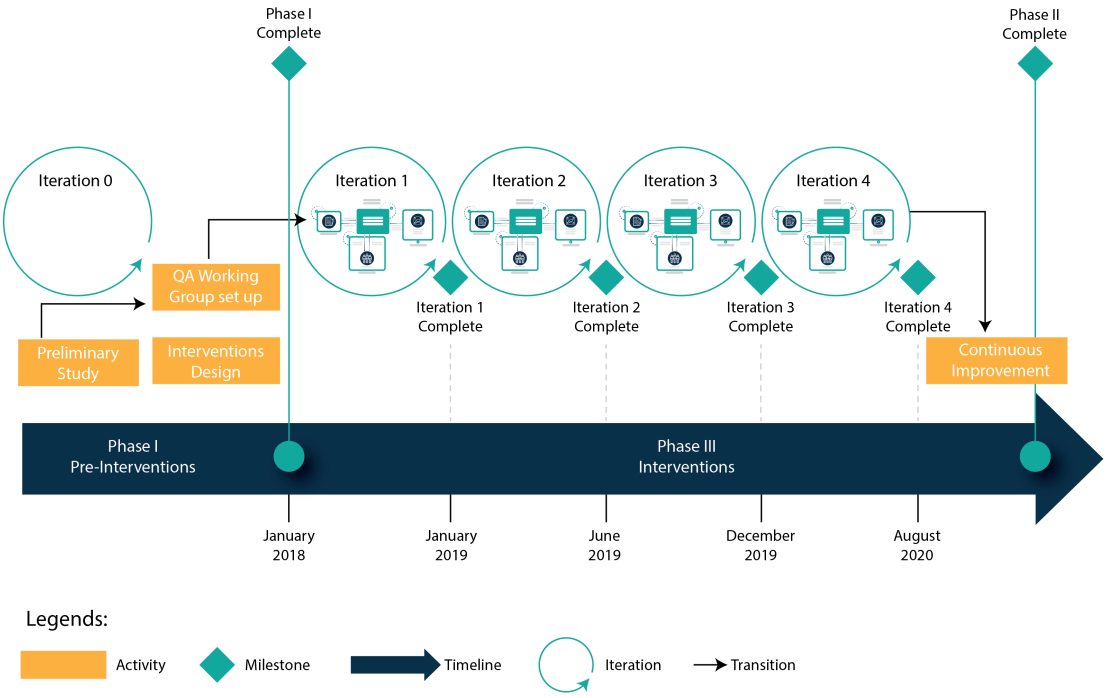

Fig. 3 ROS Community Participatory Action Research Time Line and Iterations

The PAR process (Fig. 3) typically includes two broad steps. First, research implies the participation of the community in the process of elucidating and analyzing the problems (Phase I) [20]. This process resulted in new understanding. The second phase of the research process is cyclic (Phase II). It is implemented through iterations of planning, acting, observing, and reflecting [20, 36]. We were unable to execute this phase as prescribed in the literature. In order to succeed in the community, we had to design a change process tailored for the ROS community that still embraces the principles of PAR. Section 6 presents the process we deployed to execute the implementation of interventions.

Phase I — Pre-Interventions: The process began with the identification of QA problems and issues. The objective was to discover the scope of interventions, the problem space, and the relationship between these. This was an empirical investigation of the current environment [10]. The aim was to understand the cultural characteristics of the community and the quality practices taking place in the community. This phase of the study identified the current challenges in implementing quality processes, methods, and practices in the community (see Section 5).
ROS Quality Assurance Working Group: PAR advocates for active engagement of the community [20, 36]. Participation is a democratic principle to overcome the researchers' dominance in the process and build on the knowledge of the community necessary to implement the interventions.

To this end, we established a quality assurance working group. The aim of the group is to serve as a conduit in the community for the change initiative. Its role is consultative. The first step to gain community consent for the interventions. It has been decided to implement this group as an implementation tool as it is a community cultural norm to reach consensus regarding changes in the community. The group had 23 active members. The group has been meeting monthly and has accumulated a total of 18 meetings. Minutes of each meeting were taken and published in the community forum [2].
Interventions Design: Intervention comes from the Latin intervenire, meaning "to come between, interrupt." The intention of an intervention is to make things better or to remediate a situation. In the context of this research, we define an intervention as a combination of action elements or strategies designed to produce behavior changes and improve quality assurance practice in the ROS community. Interventions may include educational programs, new or enhanced practices, improvements to the processes, or a quality promotion campaign. This research interventions create change by:

1. Influencing individuals' knowledge, attitudes, and beliefs.

[2] `https://discourse.ros.org/c/quality`

2. Increasing QA infrastructure and tools.
3. Creating supportive environments for QA practices and resources.

Interventions (see Section 6) were designed based on the findings of the phase I analysis. These interventions have been put forward to the working group for the community's deliberation and advice.

Phase II — Implementation of Interventions: The interventions have been implemented iteratively. As shown in Fig. 3, four iterative cycles have been proposed. The scope of each cycle is dictated by the interventions' priorities. The priorities have been assigned by the community working group. Members were asked to assign high, medium, or low priority for each intervention. The interventions with the highest number of high votes were assigned to the early iterations' cycles. Subsequent cycles' scope is interventions that received lesser votes for high than for medium and low priorities.

To carry out the implementation of the interventions we faced a conundrum. We had to reflect and design a process that caters for the community cultural and social specifics without disturbing the community ecosystem (see Section 6 for the proposed method).

4.3 Data Collection and Analysis

This is a qualitative research using one subject community, where data is gathered using a combination of techniques, a community working group, interviews, and participatory activities.
Data Collection. The empirical qualitative data has been captured using several techniques that were selected based on the combination of the methods and the nature of the data collected. Table 1 maps methods to their respective data collection techniques.

Methods	Data Collection Technique	Data Analysis	Key contributions
Phase I:			
Interviews & Participatory Activities	1. Ten interviews have been recorded and transcribed. 2. One researcher attended 29 community events (i.e. Conferences, meetings, etc.). Notes were taking during these events and field notes were compiled, thereafter. A total of 30 field notes have been accumulated.	Thematic Coding	1. A cultural understanding of the ROS community. 2. Definition of the problem space (see section 5.1)
Phase II:			
Participatory Activities	1. Field notes have been used to capture observations and interaction with the community. This applies to both real-life and virtual activities. Emails, and other forms of interaction with the community (i.e. Forums discussions) are captured in field notes. 2. Working group meetings and workshops have been recorded and transcribed. At this stage of the project, 18 meetings for the working group have been conducted. In addition, other ongoing meetings with volunteers working on the implementation of interventions have taken place.	Thematic Coding	1. Successful implementation of two iterations of QA interventions. 2. A change method tailored for the ROS community.

Table 1 Data Collection and Analysis Methods

Data Analysis. We used thematic coding, grounded in the interpretative philosophy. Following the guidelines of Robson & McCartan [28] and Miles, et al. [26]. The aim is to examine the meaning and symbolic content of qualitative data. The process involves the identification of themes and coding. We used line-by-line coding of the data, identifying meaningful parts of the text and assigning labels to them that indicate they are the thematic idea. This labeling or coding process enabled us to retrieve and aggregate together all the text that has been linked with a particular thematic idea. The collation of themes and association with text unveil patterns of thoughts and ideas that contribute to the understanding and later was used to formulate theories about the ROS community.

5 The ROS Community

In 2000, Stanford University began working toward integration of embodied artificial intelligence (AI), such as the Stanford AI Robot and the Personal Robots Program. They created flexible, dynamic software systems for robotic use. In 2007, Willow Garage, a visionary robotics incubator started by Scott Hassan to accelerate the advancement of non-military robotics, gave significant resources to extend and create implementations. Many researchers from various institutions and labs began to contribute time and resources to core Robot Operating System (ROS) ideas and software, all in the open BSD license. Willow Garage became a for-profit company, and ROS is maintained by the Open Source Robotics Foundation (OSRF). Over the years, the developed model has been seen as one of the strengths of ROS. This model allows any group to start their own ROS code repository that they maintain control and ownership of, and if they make their code public, they can receive recognition and credit, and they benefit from the open source software projects.

ROS is an open-source meta-operating system providing a flexible framework for writing robot software. It is a collection of tools and libraries that help to simplify creation of robotic platforms. The collaboration of researchers enables robotic products to be more robust. ROS provides a communications infrastructure as middleware. It offers asynchronous message passing, recording and playback of messages, request and response remote procedures, and a distributed parameter system. In addition to middleware communication, ROS provides robot specific features, such as standard message definitions, a robot geometry library, robot description language, diagnostics, pose estimation, localization, mapping, and navigation. The ROS toolset is one of its strongest features, and this toolset enables debugging, plotting, and visualizing the state of the system with rviz and rqt tools.

All together, over 3,000 ROS packages are available publically. These packages cover everything from proof of concept implementations to new algorithms for industrial quality drivers and capabilities. The ROS community has over 1,500 participants on the mailing list and 3,300 users on the Q&A forum, with 22,000 Wiki pages and over 30 edits per day.

Recently the community has embarked in an overhaul project called ROS 2 to re-engineer and re-architecture the current software. ROS 2 is under heavy development and attracted the participation of some heavy industry players (e.g. Amazon, Intel). The project has become the focal focus of the community. This enthusiasm had some consequences on ROS. The community attention and effort is shifted away from the current software to ROS 2.

5.1 The State of Quality Assurance in ROS

The ROS community is large and diverse, and it is different from other FOSS communities, as most ROS developers are not software engineers. The ROS community has awareness of quality, and has implemented a well-defined development process, defects management processes and tools, code review, continuous integration, unit testing, and knowledge sharing. However, the efforts for quality are constrained by participation motives, community priorities, meritocratic culture, sustainability, complexity, and adaptability [1].

Most robots are complex distributed systems, encompassing many fields of engineering. Thus, it is critical to implement a successful quality assurance management strategy. The scope of our engagement in the ROS community has been focused in creating sustainable QA practices.

Before this project, ROS was not without QA practices. It had a continuous integration infrastructure in place, an issue tracker and process of managing defects. However, they did not live up to the expectations of industrial participants. We do not imply that ROS is low quality. The collective intelligence of the community has produced a remarkable software platform that powers various industrial applications. Even though the popularity of ROS testifies for its quality, industrial stakeholders have been calling for visible and concrete community practices to raise the confidence in ROS even further. Hence the initiative of this project which is to bring ROS QA inline with other FOSS communities and software engineering best practices.

Once we became accustomed with the ROS community, we realized that its cultural fabric has to be accommodated. We needed a method because it was important to us to implement the interventions without disturbance. Executing the intervention blindly would have been naive, likely clash with the community and possibly failing.

5.2 ROS Community Cultural Traits

The ROS community over the years embraced the FOSS values, believes and norms. The ROS community is a meritocratic social system. It values transparency, freedom of choice, collaboration, self-management and innovation. We summarize its main cultural traits based on one year and a half worth of participatory observation in the community [1].

Meritocracy. Meritocracy is a social system in which success and status depend primarily on individual talents, abilities, and effort. The ROS community is a highly meritocratic system. Meritocracy is implemented via two mechanisms: the "Karma" system and reputation. The "Karma" system is a virtual rating system, where points are allocated to contributors by other community members for helping answering and resolving technical questions and challenges. The "Karma" points showcase the contributor's technical expertise and her engagement in the community. The more "Karma" points, a contributor has, the more important is hers status in the community. Contributors also have an additional "reputation" outside the "Karma" system. While the "Karma" system is an indication of the technical expertise and knowledge, reputation is the community's believes and opinions held about a member. Contributors are conscious of their reputation in the community and they thrive to maintain it.

In an early meeting with the community to set up the project infrastructure, a community member warned us: *"This is a meritocratic community. Anything you do must take in consideration this community aspect."* In a community meeting, the attendees stopped and congratulated a member of the community with high "Karma" points. One attendee said: *"Congratulations man! 40 thousand karmas. That's impressive."*

The challenge that ROS meritocracy poses to participatory action research projects is that of the researcher being new to the community and without "reputation" or "Karma". Our strategy was to enter into partnerships with community members enthusiastic about quality and with distinguished reputation in the community. The partnerships gave us leverage in initiating the project and establishing a community working group to be the owner of QA problems.

Transparency. The ROS community believes that decisions should be based on what is best for the entire community, not just a small group of developers and members. Transparency is ingrained in the community conducts. Decisions, opinions, concerns and ideas are publicized in the community forums. The aims is to increase participation and inclusiveness in decisions making.

In the first meeting of the quality assurance working group, a member of the group asserted: *"The meeting must be minuted and published in the forum. We want the community to know what we [are] doing."* Another member went further to suggest: *"The meeting should be video recorded and put up in the forum for the community."*

Transparency does not pose a direct challenge to the implementation of PAR. However, it is an important cultural trait that should be embedded in the process to ensure success. As per the community tradition, we publicized every decision, discussions and outcomes during the working group deliberations in the community quality assurance forum.

Freedom of choice. Contributors, in the ROS community, have the freedom to select the type of work they would do in the community. Tasks allocation is informal; there is no pre-defined process to assign tasks and roles.

A participant in the ROSCon 2017 conference commented about our project during a poster session: *"I'm not sure how you gonna do this? The ROS community likes to code and they participate because of the freedom they get. You may struggle to find people to join this project."*

Although this cultural trait is not a barrier for PAR, it can constrain the execution of actions. Especially, if the actions are not popular amongst contributors. For example, QA is deprioritized and consequently QA tasks are regarded mundane [1] in the ROS community,.

Collaboration. The ROS community is a collaborative social system. Collaboration is the bedrock of creative solutions and innovation in the community. Its members value joining forces to achieve goals. Community collaboration is essential to effective development of the community's product. Ideas and projects are announced in the community forums. Interested members join the discussion or volunteer to contribute to the project. Since its transition to a community structure, ROS software has been developed and grown from collaborative work. In the ROS community, initiatives are not handed down from a role

with a certain power. They are shared through collaborative channels where the initiator has equal power to the rest of the community contributors.

During the planning of our project, in a meeting with members of the community to discuss an approach to execute the project, a project member suggested to form a "task force" and start implementing our interventions. A community member opposed, he stated: *"That would be disastrous and calls for failure. This community doesn't operate like that! We need to engage the community."* Another community member concurred: *"I agree. We shouldn't bypass the community. We risk to alienate ourselves."*

This cultural trait is in line with PAR principles of democratic participation and communal reflection. However, PAR literature does not specify how to operationalize collaboration in large and distributed communities. We established a working group dedicated to oversee the work. The community working group serves as our main means of collaboration. To ensure inclusiveness of the larger community, we publicize the group meetings notes and decisions in the community QA forum.

The group members were recruited democratically. One of our collaborators, a community member with over ten years of experience in the community and high karma points and distinguished reputation, put a post in the community forum asking for volunteers to join the group. We succeeded attracting 23 members. However, the group meetings attendance has on average approximately 16 attendees.

Self-management. Another norm of the ROS community is informal self-management. The ROS community has one horizontal path of mobility, and members move from one role to another fluidly. At the center of the community are the core developers. The core team is managed by OSRF, a non-profit organization that has the responsibility of community stewardship. We entered the community with no prior exposure or engagement with the community. We worked on establishing our own reputation. We attended six community events and we showcased our work through presentations and posters. We adopted self-management in our PAR process by ensuring an equal distribution of power amongst participants.

Innovation. The ROS community has an innovative culture. The community environment cultivates and nurtures unconventional and creative thinking. It subscribes to the belief that innovation is not the province of an elite but the effort of the collective and that anyone in the community can be innovative. The energy of the community is directed toward creating new features and the depth and breadth of the ROS platform. In our earlier study of the ROS community [1], we observe that innovation drives the community and remains the centre of attention. Over time, this led to the deprioritization of QA practices.

A participant in the ROSCon 2018 conference commented, privately during the reception, after a presentation made by one of the authors of this paper. He said: *"This is a tough project to execute. Not only because people are not under the same roof, but because this community likes to innovate. They prefer working on new features rather than developing their processes."*

Our agenda is to push QA practices in the ROS community. The focal point of the community though is innovation. The QA tasks are seen as mundane. This misalignment of priorities is a challenge for our PAR implementation. PAR assumes that collaboration, authority and control facilitate the execution of actions. However, in an open source community, the implementation of actions relies on volunteers.

6 The Change Process and the Experience

In the following sections, we discuss how we tailored PAR4FOSS (Fig. 1) for the ROS community.

6.1 Tailoring of PAR4FOSS for the ROS Community

Figure 4 depicts PAR4FOSS for ROS and the infrastructure we put in place to carry out interventions. Upon the design of the interventions, we established a working group, ROS Quality Assurance Working Group, to be the problem owner and a platform for initial consultation. We started the process by a prioritization exercise. The outcome was a list of prioritized QA interventions (see Tbl. 2).

The group has been meeting on a monthly basis. The scope of each meeting was to discuss the specifics and details of the implementation of one of the interventions, according to the order of their priorities. The discussions are, then, summarized and publicized in the community quality assurance forum ([3]). This is an opportunity to engage the wider ROS community and be transparent about the group work.

[3] `https://discourse.ros.org/t/ros-quality-assurance-working-group-october-2019-meeting-notes/10891`

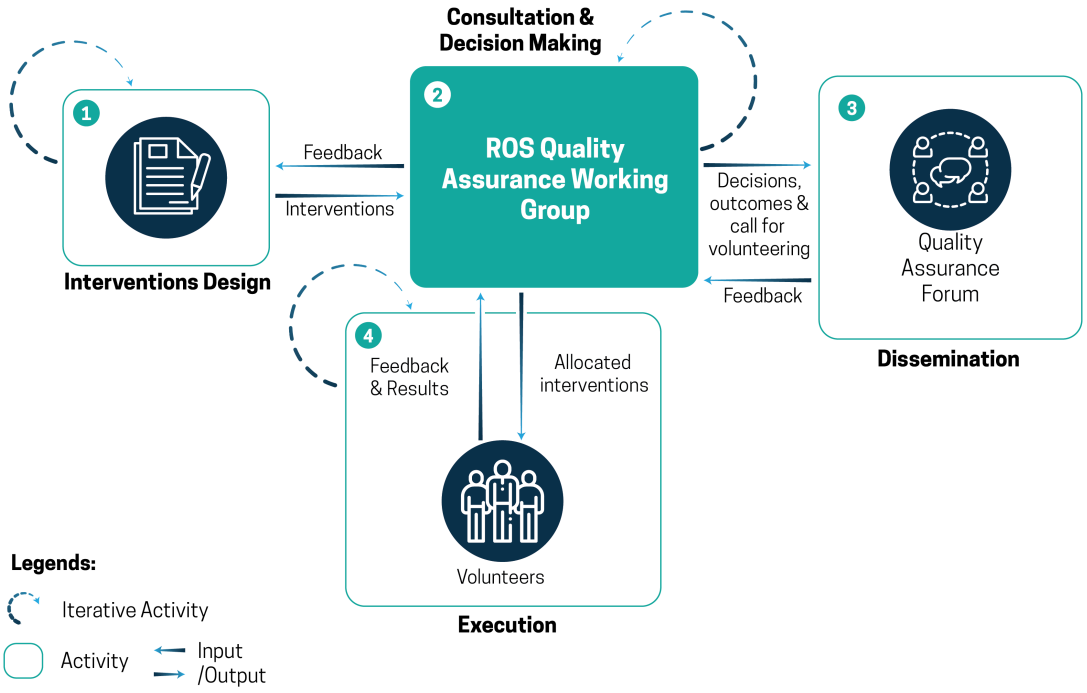

Fig. 4 PAR4FOSS Tailored for the ROS Community

Feedback then an is received from the community and communicated back to the working group for further discussion and deliberations. Once an intervention is stabilized (i.e. unlikely change in the design and the requirements of the intervention), a call for volunteering is made in the QA forum. This is the start of the execution phase (i.e. effectively implementing the interventions). The implementation of interventions is overseen by the group.

6.2 The Method

The tailored method (Fig. 4) has four dependent activities: (1) Interventions design, (2) Consultation and decision making, (3) Dissemination, and (4) Execution. These activities have inputs and outputs and some of them are iterative sub-processes. They are interlinked to constitute a method that substitutes management and authority and caters for the cultural specifics of the ROS community.

Activity 1: Intervention design. This activity is a pre-requisite to launch the implementation activities (activities 2, 3, and 4). However, the initial list of interventions has been subject of iterative review at the end of each PAR iteration (see Fig. 3), the group conducted a review session for the list of intervention. Table 2 represent the list, after one review at the end of Iteration 1.

The design of the interventions followed a systematic approach. Interventions are derived from four distinct sources: community requirements, similar communities precedents, best practices and problems (i.e. opportunities). During the inquiry phase, we identified opportunities for improvement and solutions for them. Each solution is an intervention. Interventions are composed of a set of actions (small actionable tasks). The implementation of actions will lead to full implementation of interventions. Actions are either an initiative or a corrective measure. Initiatives are new actionable changes with no prior precedent in the community. Corrective measure is a change to something existing but dysfunctional or unsuccessfully implemented. Each intervention should generate an outcome, i.e creating a QA capability and behavioral change. Failed intervention will not have an outcome and consequently no impact.

This is an iterative activity. The first version of the interventions was proposed to the working group (part of activity 2) for discussion and validation. There were no objections to the interventions proposed by the researcher. But, the group requested clarifications and definitions of some interventions. Then, the list was put in the QA forum for dissemination. Feedback was received from the wider community (part of activity 3) and discussed in the working group. Consequently, action were taken to amend the list.

Iterations	Interventions	Description
Iteration 0	Establishing a Quality Assurance Working Group	This is to establish ownership for QA problems. The QA working group has become the owner of QA.
	Quality Assurance Forum	The aim of this intervention is to create a forum for promoting dialog on quality, and the development of quality practices.
Iteration 1	Make ROS packages' quality visible (Part 1)	Create a process and tool where packages quality can be measured, assigned and displayed.
	Appoint ownership	Appoint ownership for quality assurance tools and processes.
	Energize the code review process	Code review is an existing process; unfortunately, it is loosely implemented and practiced. The aim of this intervention is to bring this practice back to ROS QA core quality practices.
	Implement a code scanning method and tool	Identify and implement state of the art code scanning and analysis techniques and tool.
	Maintenance issues	The objectives of this intervention is to attract and recruit new maintainers.
Iteration 2	Make ROS packages' quality visible (Part 2)	This is a continuation of part 1. During this part, we will develop the QA dashboard.
	Quality Hub website	A central "go-to" place for QA knowledge sharing (documentation of QA practices).
	Formalize the code ownership process	Define and implement a code ownership process. Ownership in general establishes the responsibility relationship for a software component, and a responsible developer.
	On-boarding process for core and non-core community members	Description. Define an on-boarding process for core and non-core community members.
Iteration 3	Model-in-the-loop (MIL)testing.	Identify a functional MIL testing setup that is to be used as an integral part of the CI services.
	Automated unit test generation	The goal of this intervention is to automate the creation of the tests.
	QA promotion events	Regular community campaigns to promote QA practices or support a particular community need.
Iteration 4	Model Driven Development (MDD)	Identify and demonstrate opportunities to use the paradigm of MDD for code generation.
	User Rating of packages	Implement a rating of packages feature. The ability of ROS users to rate packages quality.
	Implement a continuous improvement process	A continuous improvement process is ongoing efforts to improve quality assurance practices. These efforts are incremental review and improvement of practices in place. This is a set of community activities and processes to ensure the survival and the sustainability of quality assurance practices.

Table 2 List of iterations and their corresponding interventions

Activity 2: Consultation and Decision Making. Community consultation is a reciprocal process and a genuine partnership between the community and the researchers. This community engagement provides guidance to researchers in order to make well informed, acceptable and sustainable decisions.

This is the first activity in the implementation of an intervention (Activity 2 in the process). The first instance of consultation occurs with the working group meetings. Every meeting had a specific agenda proposed by the facilitator (i.e. the researcher) and circulated to the group members for comments, one week prior to the meeting. Then, a final agenda is put forward for the meeting. During the meeting, the facilitator endeavors to balance contributions from the attendees. Each attendee is given the opportunity to express his views.

The group initially agreed on seeking widespread or full agreement via consensus decision-making. In the process all participants' opinions are respected and their contributions are encouraged. The process in which the decision is made is as important as the resulting decision. The facilitator ensures that the power is distributed equally amongst the participants. We observed that no individual or group of individuals attempted to leverage the group to gain power. To the contrary, we noticed that the participants are highly respecting of each other opinions. No frictions in the group were observed either. Personal preferences are less important than a broader understanding of how to work together to help the community.

Decisions are put forward for voting. We use voting to reach decisions. After deliberation, decisions are put forward for voting. A simple majority is sought. We observed, irrespective of the direction a vote takes, the group accommodate the outcome and moves forward. This support for the democratic process has helped the overall process.

Activity 3: Dissemination. All decisions and the working group deliberations are publicized in the community QA forum. We ask the community to provide feedback. In some instances, we received feedback and in other instances, we did not receive any inputs beside "likes."

The aim of the dissemination is to gain a community-wide acceptance of each intervention and the decisions made by the working group. Community dissemination is communication with the wider community through the provision of relevant information and material to ensure effective consultation and transparency. The consultation and the dissemination are an opportunity to democratize the interventions and gain a broader consensus around them. This inclusiveness ensures a democratic process where community members get an opportunity to be heard and provide feedback.

The purpose of the dissemination is to democratize the interventions and gather a consensus around them prior to implementation. Interventions democratization, making interventions information available for everybody, empowers people in the community members to be part of the decision making process.

When feedback is received, it is communicated back to the working group for discussion and consideration. Although, we received some feedback throughout the process, the overall volume has been low (from no comments to ten comments). However, the views have been within the average of other ROS community forums (i.e. the highest 809). This is not a surprise. Our preliminary study [1] shows that QA does not trigger the enthusiasm of the community.

Activity 4: Execution. The execution is the actual implementation of a particular intervention. The execution commences when the intervention requirements and implementation specifications have been stabilized and there is a general positive sentiment for the intervention. This is achieved by consultation and dissemination.

The first forum, we use, to recruit volunteers to implement the actions is the working group. However, in some instances, we were not able to attract enough volunteers or to find volunteers with the right set of skills. Then, we seek volunteers in the larger community by announcing it in the QA forum.

Sub-groups have been formed in some instances to work on particular interventions (i.e. Making ROS packages quality visible). The facilitator attends all the meeting and reports the progress back to the working group.

6.3 The Infrastructure

This process necessitates a simple infrastructure to operate: an (1) online forum, (2) conferencing tool, and a (3) facilitation resource.

Online Forum. This is the platform needed to reach out to the wider community. Although, the community has already multiple forums, we opted for a dedicated forum for QA [4]. This has been seen before in other communities with well-developed QA capabilities (e.g. Mozilla and Debian). In addition of being a channel to disseminate the QA working group work and decisions, we wanted this tool to attract a sub-community interested in our agenda (enhancing the community QA practices).

Conferencing tool. The ROS community is globally distributed. Organizing meetings with attendees from three different continents has been challenging. However, the intrinsic motivation of some attendees has been a main driver. One attendee stated in a meeting *"I stay awake until 1AM to attend this meeting."*.

Facilitation Resource. The process requires a facilitator. Facilitation is the activity of ensuring the process is operational at all times. This includes organizing the QA working group meetings, facilitating the meetings, taking and publishing minutes, posting the meetings decisions and outcomes in the QA forum, etc.

[4] `https://discourse.ros.org/c/quality`

7 Evaluation of the Process

In this section, we evaluate the effectiveness of the method (Fig. 4). Effectiveness addresses the degree to which the method has been successful in producing the desired result.

The tailored method (Fig. 4) is deemed successful when its effectiveness can be demonstrated. Effectiveness is determined by (1) the method evaluation: the method activities have produced the desired results, (2) evaluation of interventions: the output (i.e. interventions implementation) is accomplished, and (3) stakeholders evaluation: are the stakeholders satisfied with the method? This can be conducted throughout the life of the project.

Collaborative action research is deemed successful when both parties (Researchers and participants) give and gain benefits, such as new knowledge or improved practical solutions [17]. Meyer explains that action research arises from a different epistemological background than other research methods. This imply, it cannot be evaluated using traditional evaluation criteria. Evaluation of action research is not necessarily based on whether change can be positively demonstrated, but instead on what was learnt from the experience of trying to change practice [25]. In the upcoming sections, we will demonstrate that our participants judged the method being useful and the method has created a change.

7.1 The Method Evaluation

The method is a set of four interlinked activities. These activities exchange inputs and produce outputs (i.e. results). In this section, we evaluate the degree to which the desired results have been produced.

Interventions Design. This activity commenced in the pre-intervention phase. The aim was to produce a preliminary list of interventions for the QA working group to assess, enhance and approve. We designed an initial list of interventions which we presented to the group for discussion and deliberation. We successfully gained a community consensus for the interventions, including by the QA working group and the wider community.

Consultation & Decision Making. This activity was implemented in the form of a consultative body, the QA working group. The QA working group has been a constructive instrument to push the QA agenda forward. The local expertise has been crucial in the review and adaptation of the QA interventions. We observed that the democratic process fits the ROS community, and the process promotes a positive organizational climate. The greater degrees of democratic openness, and transparency of governance, allowed a better engagement with the community.

The QA working group had initially attracted 23 members and attendance has been in average of 16 attendees in every meeting. However, after a year, this number dropped to seven attendees in every meeting. This did not shake the foundation of the group. The remaining attendees are the highly motivated and dedicated ones. We asked some participants to explain their absence from the group. Most participants justified their absence due to other commitments being prioritized over attending the QA working group. For example, a software engineer stopped attending the QA working group meetings. He emailed the facilitator to convey that his company management reviewed their community engagement and decided to reduce it due to other internal priorities.

Dissemination. We leveraged the QA Forum to disseminate the outcomes of consultations. We, also, leveraged community conferences to promote our work. The facilitator gave three presentations in three different community events (ROSCon 2017, ROSCon 2018, and ROS Industrial conference 2018). This has been productive. The method and the facilitator have become known in the community. Being open about the work with the method and decisions is appreciated in the community. In addition, it fits within the community transparency tenet.

Execution. ROS is a volunteer-based community. Implementing interventions requires volunteers. We attracted a modest number of volunteers to collaborate in implementing the interventions. We learned that volunteers, for this type of contributions, can be unreliable. This, because, they all have a full time employment and their commitment, to contribute to implement interventions, is during their spare time.

7.2 Evaluation of Interventions

The interventions implementation is the output of the method. Every successful implementation of an intervention is a testimony to the usefulness of the method. In this section we evaluate the nine implemented interventions and the change created with the tailored PAR4FOSS. Using the method, we executed two iterations (iteration 0 and 1) and we are half way through iteration 2. Below, we discuss the results of the interventions:

Iteration 0. Establishing a Quality Assurance Working Group. The aim of this intervention was to establish a community body to assume the ownership of QA. This was successfully implemented and the body has become a consultative structure for implementing QA interventions and a community reference for QA. In two separate instances community members reached out to the facilitator to attend the group and pitch their ideas. In another instance, the core team contacted the facilitator and requested to add a new intervention to the list, which is the migration of the current community Wiki to a new upgraded instance.

Quality Assurance Forum: The aim of this intervention was to create a focal place for QA collaboration and a channel to disseminate the QA working group's decisions. This was successfully established and became an active forum for QA-related discussions. The forum receives a steady flow of contributions within the topic of QA. It also created an awareness and visibility for the QA agenda in ROS. The official statistics reveal that in September 2017 the forum had 365 active users and 441 posts.

Iteration 1. Make ROS packages' quality visible (Part 1). The analyses conducted in the pre-intervention phase of PAR and the QA working group identified the lack of visibility into the existing test results as one of the main issues to address. The outcome of the first few meetings of the QA group underlined the importance of reporting Continuous Integration (CI) infrastructure test results. Prior to this intervention, the results of various tests and checks were hidden behind interfaces that require numerous clicks to navigate through. This intervention consolidated the CI test results into a badge (Fig. 5 where the results can be accessed by a simple click. A drop-down menu provides access to a configurable number of historical results, which allow developers to assess a package's health based on trends rather than rely on just the most recent results. This enhancement has produced the following results:

1. We gave ROS wiki visitors insights into the status of tests run by the CI system for a particular ROS package.
2. We provided ROS wiki visitors with a direct way to access package status by consulting test results via UI elements that do not require additional clicks.
3. We enabled ROS wiki visitors to access, by a single-click, the test results stored in the CI system.
4. We provided ROS wiki visitors with the ability to view historical trends in testing results for a particular ROS package.
5. We introduced an element of gamification (i.e. reward developers by showing green badges, checkmarks and other achievement in the form of UI elements). Packages for which all tests are succeeding are rewarded with a green badge with a checkmark. This is the desired situation for all packages.

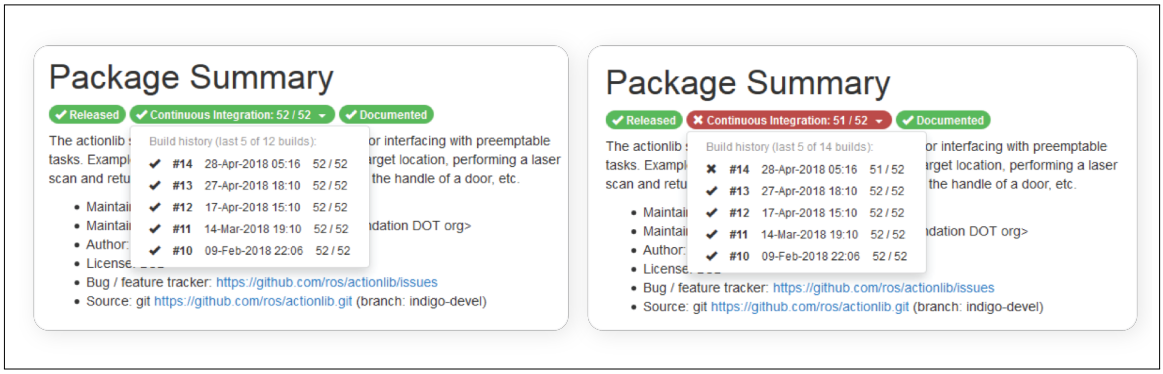

Fig. 5 Green (all tests passed) and Red badge (not all tests passed) and drop-down displaying build history

Appoint ownership. The aim of this intervention is to appoint ownership for ROS quality assurance processes, tools and infrastructure. The QA working group agreed to appoint a default owner to each QA

initiative. A default owner is the individual, group of individuals or institution who worked to implement the initiative. If the default owner objects to the ownership, then the QA working group will look in the community to recruit an owner. Ownership was originally suggested to safeguard the sustainability and to ensure the continuous improvement of these interventions. We successfully appointed owners to the implemented interventions.

Energize the code review process. ROS has introduced code review to its quality assurance practices, but this could not be sustained due to various influencers (e.g. prioritizing innovation over quality). The aim of this intervention was to this bring back to ROS. The QA working group selected three ROS repositories, ros-comm, rviz and MoveIt, to pilot code review, before an incremental roll out to the rest of ROS repositories and projects. Disappointingly, we learned that the core team has pulled out resources from ros-comm and rviz maintenance due to shifting focus to work on the ROS2 project. For MoveIt, we observed that only maintainers review code which strains the maintenance resources in the project. We approached eight contributors to investigate why they did not review code. We learned that the most common reason was lack of expertise. One contributor stated: *"It took me a long time to feel like I understood MoveIt enough to contribute, and it definitely takes longer to be able to understand someone else's contribution, especially if it's in a part of the code that you aren't familiar with. MoveIt is a very big project, so this happens pretty frequently: personally, I don't feel like I have enough knowledge to review anything outside of MoveIt Core and the OMPL interface, and don't do so often."*

Consequently, the QA working group decided to pilot code review in the ROS2 project where there is enthusiasm and resources. We currently negotiate with the ROS2 team the implementation of the pilot. This intervention will be cycles of trial, errors and learning before it will be implemented successfully. ROS is a complex ecosystem and its technology is not beginner friendly either. This constrains the participation in code review, only experienced maintainers and contributor can review code. This will have some implications at the community level. ROS may need to consider lowering its entry barriers to attract a steady flow of newcomers. During the discussion of this intervention in the QA working group a participant stated: *"we [ROS community] have to rethink our participation model, if we want to sustain these practices [QA practices]. Currently, newcomers face a complex ecosystem to understand, before they can contribute. This chases away newcomers and impacts our sustainability."*

Even though some members start thinking that some of the interventions are beyond what the group is able to deliver, the majority of the group members are determined to persist. At this stage, this intervention is a work in progress. But, it already demonstrates signs of success. The method has shown the ability to adapt to its environment constraints and obstacles.

Implement a code scanning method and tool: This intervention is executed by researchers. Two angles of attack were considered. First, we have studies quality challenges faced by ROS users and contributors as observed in the issue tracking system of several packages from the distribution. Based on this analysis, we agreed to develop several project specific linters that can detect these ROS-specific issues (mostly build-time, load-time, and run-time dependencies of various kinds). While the checkers are fairly easy to build, the community still needs to work on how to embed them in the code-review and approval process. Second, we have used seed funding available to sponsor a one-time dedicated security scan of ROS and ROS2 packages performed by Alias Robotics. This work resulted in proposing a method of scanning ROS and ROS2 packages in an effort to create a secure codebase.

Maintenance. ROS is suffering from orphaned packages and lack of volunteers to maintain code. Researchers proposed to design and implement a funding model for unfunded maintenance activities. However, this proposal was strongly rejected by the group. A participant commented, *"we are not bringing money in. That's not what we are here for."* This view was strongly supported by other group members. This issue was discussed in-depth during two consecutive sessions. The QA working group concluded that this is a community problem that cannot be addressed by the effort of the QA working group. There was a consensus to de-scope the intervention. Although this can be classified as a failed intervention, the experience has some merits. The group has demonstrated its ability to critically evaluate the interventions and make practical judgments about their faiths. The group concluded that addressing this issue is beyond its mandate. The problem is deeply rooted in the community make-up. ROS is a complex ecosystem. The learning curve for newcomers is long and difficult. Consequently, the community does not attract a steady flow of newcomers, which strains its participation model. This cascades to all aspect of participation in the community, including maintenance.

Iteration 2. Making ROS packages quality visible (Part 2). This is the second phase of making ROS packages quality visible. The intent of this intervention is to roll out a "Quality Dashboard" for every ROS and ROS-Industrial package. The QA working group spent multiple session defining and designing

the quality dashboard. The group defined a set of software engineering quality metrics [5] [6] (e.g. Unit test coverage, code style violations, number of closed issues, etc.). Haros, a framework for static analysis of ROS-based code, was extended by volunteers to generate these metrics. This will be integrated into the ROS BuildFarm to generate an output file, which will be presented in a Wiki UI for ROS developers to consult, similarly to the CI Badge (Part 1 of the intervention). This will enhance further the visibility of packages' quality. This work is in progress.

Quality Hub website. ROS Quality Hub is an online community knowledge base. It is a go-to-place for QA knowledge (e.g. articles, tutorials, etc.). The content will be volunteered contributions from community members. The website has been setup. The QA working group compiled the initial content. The group is yet to define a strategy to encourage the wider community to contribute.

7.3 Stakeholders Evaluation

This evaluation sought stakeholders assessment of the method and its facilitation. To this end we organized a reflection session. After one year, the QA working group dedicated one of its meetings to a reflection on the group work and the method.

The attendees were asked to provide feedback on the method and its facilitation. We observed that the community portrays the method as the facilitator. Over time, the facilitator personifies the method. This association was reflected in the feedback received from the QA working group participants.

The feedback was overwhelmingly positive. In regards to the method, attendees expressed their satisfaction with its implementation and fitness for the ROS community. One participant said: *"The process works well for the community. But we cannot fix every problem! They [the community] can't except us to fix everything. There are unfixable things and we need to admit it."* Another participant echoed the previous opinion. He stated: *"I think the process works fine for the community. You've done a great job ... but we need to divide the work into actionable items that we can achieve in a month. If we commit ourselves every month to small achievable items, we will move faster and achieve more."*

We have not noticed any resentments or objections to our work or the method. To the contrary, the community is appreciative of the method and our achievements. The facilitator occasionally receives emails from the community members praising the work being done. In one email from an engineer, he stated: *"I appreciate the work you do for the community. I wish, I could help. Unfortunately, our management restricted our involvement in the community to certain tasks. I wish you luck in these initiatives you taken."*

8 Discussion

Action research is designed for conventional organizations. We proved that its adaptability for a FOSS community and applicability to software engineering. The primary novelty of this work (i.e. the PAR4FOSS method) is the ability to introduce change to a FOSS community. The method has proven to be successful in the ROS community. To the best of our knowledge, an adaptation of PAR for FOSS communities has not been attempted previously. The difficulties experienced currently are a natural manifestation of a method in practice. Once a method is deployed in the real world, it would face obstacles and suffer from the influence of various forces.

Given this is a process improvement endeavor, we considered software process improvement (SPI) as an alternative. However, this proposition was dismissed. SPI mandates change in a less participatory style. We wanted a method that engages the community and leverages local knowledge, which we did not have at the beginning.

We now reflect on the main challenges and difficulties observed when working with the proposed method.

The QA working group is at the core of the method, leveraged for consultation and decision making, it is crucial to sustain the participation. Attendance declined due to other priorities, conditions and motivation. This issue was put forward for discussion in a meeting. The group decided to work on the motivation to participate. It has been suggested to change the meeting structure. Instead of dedicating the whole hour to discussing the interventions, the group suggested to dedicate the first 30 minutes for

[5] `https://discourse.ros.org/t/ros-quality-assurance-working-group-meeting-minutes-april-2018-meeting/4473`

[6] `https://docs.google.com/spreadsheets/d/1Ujwc2rjmywWpiamCGNRAdD3USNurXLqunIvgSrSbwvM/edit#gid=0`

a presentation by guests, experts and contributors on a relevant topic to the group, followed by a ten minutes discussion. Then, the remaining 20 minutes is dedicated to interventions implementation. We recently implemented this new format; we are yet to learn how it enhances the participation in the QA working group.

Lesson 1: Competing priorities hinder the motivation to participate in the process (i.e. participation in the QA working group) by the community contributors. Motivation to participate should be nurtured throughout the process.

We learned that volunteers can be unreliable. The volunteers we recruited have full time jobs. Their involvement in the project takes place in their limited spare time. This has decelerated the execution of interventions and may have an impact on the time line of interventions' implementation. We recognize that these are the means available in the community. The method operates with the resources available in the community and it inherits the community constraints and conditions. The interventions need to be regulated and tailored to the available resources.

Lesson 2: Most volunteers in the ROS community are affiliated. They have full time employment and their commitment to the project is during their spare time. This constrains the implementation of interventions.

We learned that the whole method should remain flexible and adaptive. The community is fluid and its priorities keep changing. Few months after we started our project, the community launched the ROS2 project, a dramatic re-architecturing of the current ROS platform. ROS2 is meant to address some of the fundamental technical design issues found over time in ROS. Upon the completion of ROS2, it will be up to each ROS user to decide whether to migrate their code to ROS2. ROS2 has attracted the attention and the enthusiasm of the community. The core team has shifted most of its resources to the development and the maintenance of ROS2. The project also attracted the participation of large and influential companies (e.g. Intel, Amazon, and Bosch). This shift of priority has consequences on the portfolio of interventions. The QA working group is facing a challenging question, whether to re-design the interventions to reflect the community focus on ROS2 or remain as is. The group does not seem to have the urgency to shift the focus toward ROS2, because interventions are universal, and can apply to both ROS and ROS2. However, the enthusiasm for ROS2 has impacted the project. For example, one of the attendees of the QA working group, a member of the core team, has stopped attending the group meetings. When asked about the reasons behind his absenteeism, he replied *"we were instructed to not spend time and effort outside ROS2."* This does not have a direct impact on the design of the method. The portfolio of interventions can be refreshed to reflect the focus on ROS2 if needed. But, this shows that the method in use has to be adaptable to its changing environment.

Lesson 3: The method should be flexible and willing to adapt to its changing environment. This quality ensures the method will continue to deliver.

We learned that managing resources for an open source project is a challenge. Some of our interventions require advanced skills in the community software tools and infrastructure. Finding available volunteers with such skill is difficult. Even though we managed to secure the participation of a contributor with in-depth knowledge of the community tools and infrastructure, his commitment to the project has been fluctuating.

Lesson 4: Securing the participation of resources with the right skills is a challenge. FOSS projects should ensure a steady participation from contributors with skills matching the needs of interventions.

The QA working group occasionally reflects upon these lessons. While this endeavor is a delivery process, it is at the same time a learning journey.

Limitations We identified some limitations of this study. First, the method relies heavily on the facilitator to ensure the operations of the method. This role can be undertaken by a change enthusiast in the community. Second, the interventions design requires a skilled analyst or researcher. Not all FOSS communities can warrant the presence of a researcher to conduct the analysis required to compile a portfolio of interventions. However, this is an inherent feature of PAR. Third, the commitment to the

working group can fluctuate. Members of the group have left the group, due to their other working obligations and new members joined half way through. This fluctuation disturbs the process. Finally, the group priorities may change in a long four year project. This will compromise the execution of the designed interventions.

This method is evaluated in one FOSS community. Ideally, for a method to mature, it needs to go through multiple evaluations and ongoing improvements. We hope this work would inspire other action researchers to implement, evaluate and improve the method in other FOSS communities. We also intend to commence similar projects in others communities.

We have only used the method in a FOSS community. FOSS communities are a type of self-managed teams. Thus, we believe this method can be abstracted to cover other self-managed organizations and teams. The aim of the abstraction is to offer a method that can be instantiated in different circumstances and communities that share similar cultural traits to ROS.

9 Conclusion

We initiated a project in the ROS community with the objective to enhance the current quality assurance practices in the community. We opted for the native form of PAR but we quickly stumbled. We reflected upon the situation and we draw conclusions. These conclusions shaped the process (PAR4FOSS) we put in place. According to Mathiassen, uncertainty, instability, uniqueness, and contradiction exist in action research [24]. Reflection and learning are key elements in action research, and researchers must open their mind to engage in reflections, dialogues, and research efforts [24].

We proposed a participatory action research method tailored specifically for the ROS community. We have found that the ROS community shares its core values, believes and culture with other FOSS communities. Hence, we suggest that the proposed method can be leveraged to introduce change in other FOSS communities.

The strengths and the value of PAR4FOSS is that it provides a powerful means of improving and enhancing practice. It also untackles complexities of the situation. What we proposed is a first building block in a long journey to build a method for introducing change to FOSS communities. Methods building requires experimentation and errors. Learning from the errors strengthen the method and push toward maturity. We hope that others will be able to build on this experience.

Acknowledgments

This work is supported by the EU's H2020 research and innovation programme, grant No 732287 ROSIN. We thank the volunteers for their work in the implementation of interventions. We thank the ROS quality assurance working group members for their ongoing inputs, advice and participation in the process.

References

1. Adam Alami, Yvonne Dittrich, and Andrzej Wąsowski. Influencers of quality assurance in an open source community. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '18, New York, NY, USA, 2018. ACM.
2. Gary Anthes. Open source software no longer optional. *Communications of the ACM*, 59(8), 2016.
3. Richard Baskerville and A Trevor Wood-Harper. Diversity in information systems action research methods. *European Journal of information systems*, 7(2), 1998.
4. Richard L Baskerville. Investigating information systems with action research. *Communications of the AIS*, 2(3es):4, 1999.
5. Jürgen Bitzer, Wolfram Schrettl, and Philipp J. H. Schröder. Intrinsic motivation in open source software development. *Journal of Comparative Economics*, 35(1), 2007.
6. David Bretthauer. Open source software: A history. *Information Technology and Libraries*, 21(1), 2002.
7. Martin Campbell-Kelly. Historical reflections Will the future of software be open source? *Communications of the ACM*, 51(10), 2008.
8. Andrea Capiluppi and Karl Beecher. Structural complexity and decay in floss systems: An inter-repository study. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*. IEEE, 2009.
9. Paul A. David, Andrew Waterman, and Seema Arora. FLOSS-US the free/libre/open source software survey for 2003. *Stanford Institute for Economic Policy Research, Stanford University, Stanford, CA (http://www. stanford. edu/group/floss-us/report/FLOSS-US-Report. pdf)*, 2003.
10. Yvonne Dittrich, Kari Rönkkö, Jeanette Eriksson, Christina Hansson, and Olle Lindeberg. Cooperative method development. *Empirical Software Engineering*, 13(3), 2008.
11. Margaret S Elliott and Walt Scacchi. Mobilization of software developers: the free software movement. *Information Technology & People*, 21(1), 2008.

12. Brian Fitzgerald. The transformation of open source software. *Mis Quarterly*, 2006.
13. Daniel M German. The gnome project: a case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201–215, 2003.
14. Rishab A. Ghosh. Understanding free software developers: Findings from the FLOSS study. *Perspectives on free and open source software*, 2005.
15. Rishab A. Ghosh, Ruediger Glott, Bernhard Krieger, and Gregorio Robles. Free/libre and open source software: Survey and study, 2002.
16. Alexander Hars and Shaosong Ou. Working for free? Motivations of participating in open source projects. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE, 2001.
17. Robert Home and Niels Rump. Evaluation of a multi-case participatory action research project: The case of solinsa. *The Journal of Agricultural Education and Extension*, 21(1):73–89, 2015.
18. Chris Jensen and Walt Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007.
19. Stephen Kemmis and Robin McTaggart. *Participatory Action Research: Communicative Action and the Public Sphere*. Sage Publications Ltd, 2005.
20. Stephen Kemmis, Robin McTaggart, and Rhonda Nixon. *The action research planner: Doing critical participatory action research*. Springer Science & Business Media, 2013.
21. Bruce Kogut and Anca Metiu. Open-source software development and distributed innovation. *Oxford review of economic policy*, 17(2):248–264, 2001.
22. Martin F Krafft. *A Delphi study of the influences on innovation adoption and process evolution in a large open source project: the case of Debian*. PhD thesis, University of Limerick, 2010.
23. Karim R. Lakhani, Robert G. Wolf, and Others. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Perspectives on free and open source software*, 1, 2005.
24. Lars Mathiassen. Collaborative practice research. *Information Technology & People*, 15(4):321–345, 2002.
25. Julienne Meyer. Evaluating action research. *Age and ageing*, 29(suppl_2):8–10, 2000.
26. Matthew B Miles, A Michael Huberman, Michael A Huberman, and Michael Huberman. *Qualitative data analysis: An expanded sourcebook*. sage, 1994.
27. Sinan C. Özbek. *Introducing innovations into Open Source projects*. PhD thesis, Freie Universität Berlin, 2011.
28. Colin Robson and Kieran McCartan. *Real world research*. John Wiley & Sons, 2016.
29. Richard M. Ryan and Edward L. Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1), 2000.
30. Richard M. Ryan and Edward L. Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55(1), 2000.
31. Walt Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings-Software*, 149(1), 2002.
32. Walt Scacchi. Free and open source development practices in the game community. *IEEE software*, 21(1), 2004.
33. Walt Scacchi. Socio-technical interaction networks in free/open source software development processes. In *Software Process Modeling*. Springer, 2005.
34. Walt Scacchi. Free/open source software development: Recent research results and methods. *Advances in Computers*, 69, 2007.
35. Maureen A Scully. Meritocracy. *Wiley Encyclopedia of Management*, pages 1–2, 2015.
36. Ernest T Stringer. *Action research*. Sage publications, 2013.

E

Appendix E: Paper E

How Do FOSS Communities Decide to Accept Pull Requests?

Adam Alami
IT University of Copenhagen
Copenhagen, Denmark

Marisa Leavitt Cohn
IT University of Copenhagen
Copenhagen, Denmark

Andrzej Wąsowski
IT University of Copenhagen
Copenhagen, Denmark

ABSTRACT

Pull requests are a method to facilitate review and management of contribution in distributed software development. Software developers author commits, and present them in a pull request to be inspected by maintainers and reviewers. The success and sustainability of communities depends on ongoing contributions, but rejections decrease motivation of contributors. We carried out a a qualitative study to understand the mechanisms of evaluating PRs in open source software (FOSS) communities from developers and maintainers perspective. We interviewed 30 participants from five different FOSS communities. The data shows that acceptance of contributions depends not only on technical criteria, but also significantly on social and strategic aspects. This paper identifies three PR governance styles found in the studied communities: (1) protective, (2) equitable and (3) lenient. Each one of these styles has its particularities. While the protective style values trustworthiness and reliability of the contributor, the lenient style believes in creating a positive and welcoming environment where contributors are mentored to evolve contributions until they meet the community standards. Despite the differences, these governance styles have a commonality, they all safeguard the quality of the software.

CCS CONCEPTS

• **Software and its engineering** → **Open source model**; • **Human-centered computing** → Collaborative and social computing.

KEYWORDS

Open source software, code review, pull request, decision making, FOSS governance, community management

ACM Reference Format:
Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2020. How Do FOSS Communities Decide to Accept Pull Requests?. In *Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383219.3383242

1 INTRODUCTION

FOSS projects are collaborative ventures organized as communities that produce software using specific coding processes and tools such as GitHub. Contributors submit code changes, such as a bug fix or a new feature, in form of a pull request which undergoes an evaluation for appropriateness and quality. The evaluation process is not simple; it involves specific technical and social rituals. Various evaluation patterns (governance styles) emerge in communities. Only 13% of pull requests are rejected due to technical reasons [15] and "the toughest and most frequent challenges encountered by contributors are social in nature." [16]. We attempted to determine the factors affecting pull requests being accepted or rejected by asking the following questions:

RQ1: *How do FOSS communities decide to accept pull requests?*

RQ2: *What are the principles of evaluating pull requests?*

We define PR governance as the system of rules, practices, and norms by which a community directs and control assessment of PRs. It ensures management of the interests of the community and the integrity of its products. Because of the importance of PRs and the effect the governance of FOSS communities has on PRs, this study was designed to investigate PR evaluation governance styles in FOSS communities, using a qualitative approach. We determine the decision-making mechanisms in evaluating PRs based on extensive data transcribed from interviewing 30 FOSS contributors and maintainers from five communities.

The PR evaluation process is a socially loaded practice. Interviewee 25 explains: *"... how contributions are rejected is a major factor in a project's success. The structure of PRs acceptance process is such that it can easily be used to bully people, assert dominance, engage in various forms of emotionally abusive behavior"*. If project's success depends greatly on how PRs are evaluated; then, it is important to understand how communities evaluate PRs.

In this work we identify existing PR governance styles and their underlying believes and norms. We also extract lessons for community leaders and maintainers. For example, our interviewees prefer a governance style that values technical merits over social connections. We highlight the main contributions below.

(1) In response to **RQ1** we propose to distinguish the following three governance models:
   (a) *Protective*: A defensive style of governance where the project leader and his subordinates have absolute power over what is merged into the code. This is known in the FOSS circles as "no by default." This attitude requires prior commitment and trust from the contributor to win the approval of the gatekeeper before the evaluation can take place.
   (b) *Equitable*: A style of governance based on fairness and the ascendancy of evaluation principles. It focuses on a balanced and technically grounded decision. The community principles overrule any leniency toward contributors.
   (c) *Lenient*: A style of governance based on creating a positive and welcoming environment for contributors. This style tolerates some errors and mediocrity. The foundational belief here is that a contribution is an asset that should

EASE 2020, April 15–17, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-7731-7/20/04…$15.00
https://doi.org/10.1145/3383219.3383242

not be taken lightly. A contributor carries an enthusiasm that should be leveraged for the benefit of the community. In order not to compromise on quality, the contributor is guided to evolve his or her PR to mergeable quality through mentoring. Our data shows, despite their fundamental differences in governing code changes, these style of governance have one thing in common, safeguarding code quality and ensuring the evolution of the software as per the community roadmap.

(2) In response to **RQ2**, we identified criteria that fit into three categories: (1) Software engineering practices and requirements, (2) social norms, and (3) strategic vision for the product. First, the software engineering criteria include specific rules and recommendations, which contributors need to adhere to in order for their PRs to be of "mergeable" quality. Atomicity is an example. The studied communities require that a PR addresses a single atomic concern. Second, social community norms, like trustworthiness, guide behavior of developers. A contribution from a trustworthy community member, who demonstrated prior commitment, might be prioritized. Third, the product vision must be met, or at least not contradicted, by the intent of the contribution.

FOSS communities are unique social systems and their PRs governance models will reflect this uniqueness. Understanding the values, norms and rituals that are taken into consideration in the PRs evaluation governance, enhances our insight into the evolution, growth, and lifecycle of open source software communities.

To best of our knowledge, all key contributions of this paper (the governance models and the analysis of social and technical interactions in pull request evaluations) are entirely new. We revealed how communities collectively judge and decide on the fate of a code change submission; while previous work focused on chances of acceptance and the factors affecting acceptance and rejections [16, 17, 23, 33, 34, 40].

The paper proceeds by introducing the studied communities in Sect. 2. In Sect. 3, we define our research method and discuss the rationale behind it. Section 4 presents the key findings, and Sect. 5 interprets them. Related work is discussed in Sect. 6. We conclude in Sect. 7.

2 SUBJECT COMMUNITIES

We now present the studied communities. We intentionally sought diversity in our selection. Our subjects build different products, have different participation demographics and history. Each community was selected for a particular reason. FOSSASIA was selected for having a predominately southeast Asian contributors and having an agenda beyond software development. Odoo and DuckDuckGo were selected because both of them are a blend of a community and a commercial enterprise. The Coala community is relatively young (5 years old at the time of the study), but has shown strong signs of growth. While the above communities are rarely studied, many have investigated the Linux Kernel community. Previous studies of the community [2, 20] have shown that the community has its own particularities. Being different culturally and socially from other communities, inspired us to include the Linux Kernel in our selection of communities. Table 1 summarizes the selected communities and the dimensions used for the selection.

FOSSASIA. FOSSASIA was founded in 2009, as a community committed to creating "social change" by using a wide range of technologies. The community projects range from free and open source software, to design, graphics and hardware. Amongst its successful projects, SUSI.AI and EventYaY. SUSI.AI is an artificial intelligent application that provides functionality for personal assistance, Help Desks and Chatbots. EventYaY offers features for organizers to create and manage events.

FOSSASIA welcomes developers to contribute using Github. The PRs management process uses the default Github features of managing PRs. Changes, new features and bug fixes are raised as "issues" and discussed by the community. Contributors voluntarily select and work on issues. Once the code and its companion artifacts (i.e. Test, documentation, etc.) are available, the contributor raise a PR to resolve the issue. The PR is assessed using a set of technical and non-technical criteria.

Odoo. Founded in 2005, Odoo is both a company and a community. The community develops software to manage and record sales, inventory, procurement, and accounting functions: a business intelligence engine with an all-in-one business suite program. The company markets an "enterprise" version of the community platform. The Community version is the open source version. The company version supplements the community version with additional features and services. The community attract developers worldwide.

DuckDuckGo. Similarly to Odoo, DuckDuckGo is a company and a community. Developing an independent search engine and web browser. The company opened source the software to attract a community around it; but their indexing algorithm remains closed source. DuckDuckGo was founded in February 2008. By 2013, there were over 3 million users on DuckDuckGo. In 2014, DuckDuckGo was included in Safari, and it was built into Mozilla.

Coala. Coala is a tool for developers made by developers, a language-independent Linter and analysis toolkit. The primary goal of coala is to make it easier for developers to create rules for a project's code. Coala emphasizes reusing code and the ability to use plugins. Coala provides a unified interface for linting and fixing code with a single configuration file. The software was first released in July 2015. Six additional releases were made in 2016, and two additional versions were released in 2017. The community positions itself as a beginner friendly environment. Coala outlines exactly how to get started as a member of the community, and the first step to meet the community.

Linux Kernel. The Linux Kernel is a free and open source operating system. Now, Linux is the most popular and versatile operating system kernel. It is used on super computers and web-servers, powering up cloud infrastructure, and controlling lots of mobile and embedded devices including all Android devices.

Communities	Dimensions			
	Product(s)	Size (estimated # of contributors in GitHub)	Age (years)	Leadership
FOSSASIA	The Open Event Organizer, SUSI.AI, PSLab Android App, and NeuroLab Android	1500	10	Meritocratic system
Odoo	Enterprise resource planning (ERP)	1000	15	Meritocratic system
DuckDuckGo	Search engine	186	11	Meritocratic system
Linux Kernel	Operating system kernel	15,600 [13]	25	"Benevolent dictator" [2]
Coala	Language independent linter	437 contributors	5	Meritocratic system

Table 1: Selected Communities and the Dimensions used for Selection

3 METHODS

We want to understand the process that takes place in the assessment of PRs in FOSS communities; the what and how of a PR evaluation and decisions are taken to merge or reject PRs. We wanted to explore the human and social aspects of the PR evaluation process. For this reason, we choose a qualitative research method, suitable for exposing and gaining participants' experiences and perspectives, giving rich data. This allows us to gain breadth and depth of understanding. We conducted interviews with contributors and maintainers from five communities, (see section 2).

Interviews. We opted to use semi-structured interviews as they allow the researcher to add questions arising during the interview. A semi-structured interview is suitable for assembling rich data for a qualitative study, as qualitative studies explore topics with a goal of gaining insights into individual beliefs and behaviors.

The questions (Tbl. 2) in our interview fall into three categories: introductory, core, and probing questions. The introductory questions were used to set the tone for the interview and make the interviewee comfortable. The core questions are directly related to the research questions. The probing questions are aimed at exposing details and concrete facts.

intro	Can you talk to me about your community?
	What first motivated you to participate in this community?
core	Can you describe the PR evaluation process in your community?
	Can you talk to us about the experience of having a PR rejected?
	Can you talk to us about the experience of having a PR accepted?
	When you evaluate a PR, how do you go about it?
	What is your community attitude and philosophy regarding evaluating PRs?
probing	What were the reasons for rejecting your PR?
	How did you feel about the rejection?
	What were the reasons for accepting your PR?
	How did you feel about the acceptance?
	What is the maintainer role in the process?

Table 2: Key parts of the interview framework

Subject Selection. We selected five FOSS communities that we felt would give us a deep understanding of the phenomena under study. Section 2 summarizes the choice and the selection process of the communities. We interviewed 30 participants from FOSSASIA, Odoo, DuckDuckGo, Linux and Coala communities. We searched for participants on GitHub repositories with the exception of the Linux and FOSSASIA communities. We randomly (indiscriminately, without a method, or conscious decision) searched for contributors and maintainers with valid emails in their GitHub profiles. Then, we sent them an invite to participate in the study. For the Linux and FOSSASIA communities, we used our contacts in the community to recruit participants. A snowball sampling effect took place in the recruitment of participants. We asked our contacts to introduce us to contributors and maintainers for the purpose of this study. Table 3 summarizes the demographics of the population of the interviewees. The experience is the number of years the interviewee spent in contributing to open source. Maintainers have final responsibility to merge the code and ensure an adequate review has occurred before the merge. They also direct the contributors and reviewers,

Interviewee	Community	Role	Experience [Y]	Country
1	FOSSASIA	Maintainer	4	India
2	FOSSASIA	Maintainer	5	India
3	FOSSASIA	Maintainer	4	India
4	FOSSASIA	Contributor	3	India
5	FOSSASIA	Maintainer	4	India
6	Odoo	Contributor	10	India
7	Odoo	Contributor	10	Greece
8	Odoo	Contributor	12	Belgium
9	Odoo	Contributor	3	Italy
10	Odoo	Contributor	5	India
11	Odoo	Contributor	8	USA
12	Odoo	Contributor	15	Belgium
13	DuckDuckGo	Contributor	6	USA
14	DuckDuckGo	Contributor	8	UK
15	DuckDuckGo	Contributor	5	North Macedonia
16	DuckDuckGo	Contributor	11	India
17	DuckDuckGo	Maintainer	12	USA
18	DuckDuckGo	Contributor	9	Finland
19	DuckDuckGo	Contributor	3	India
20	Linux Kernel	Contributor	12	Finland
21	Linux Kernel	Contributor	10	USA
22	Linux Kernel	Contributor	5	Ukraine
23	Linux Kernel	Contributor	6	India
24	Linux Kernel	Maintainer	8	North Macedonia
25	Linux Kernel	Contributor	30	USA
26	Coala	Contributor	5	India
27	Coala	Contributor	4	South Korea
28	Coala	Contributor	6	India
29	Coala	Maintainer	4	India
30	Coala	Maintainer	6	India

Table 3: The population of the interviewees

making sure that they connect to each other appropriately, often serving as dispatcher. Contributors are developers and sometimes volunteer to review other developers' code.

Data Collection. As the subjects were distributed geographically, all interviews were conducted using Google Meet (a video conferencing tool). The interviews lasted from 40 minutes to an hour, and they generated, in average, 12 pages of verbatim. All interviews were transcribed from recorded interviews.

Analysis. We used thematic coding [6, 14] to analyze the data, following the guidelines of Robson and McCartan [30] and of Miles and coauthors [24]. The iterative analysis begun in the early stages of the data collection and continued throughout the study. The responses were coded by examining the data line-by-line through the lens of the following questions: what is this saying? What does it represent? What is happening here? What are they trying to convey? What is the process being described? Once the responses were coded, we could find patterns in statements and ideas that were then suggestive of a theme (i.e. a concept or implied topic that organizes a group of repeating ideas that help to understand the responses related to the research question). After identifying and giving names to the basic meaning units, we grouped them in categories by similarity. Table 4 shows examples of our codes and their categories.

We stopped conducting interviews, when we attained saturation so when (1) all the data are accounted for, with no outlying codes or categories; (2) every category is sufficiently explained in depth by the data that support it; and (3) there is enough data to ensure the research questions can be answered.

4 FINDINGS

4.1 RQ1: Decision Making in PR Evaluation

We identify three styles of governance for pull requests in our data: (1) protective, (2) equitable, and (3) lenient. Each of these styles has certain characteristics and qualities. Table 5 represents the studied communities governance styles.

4.1.1 Protective. This style is defensive; it values trust, relationships and reliability of the contributor. The Linux Kernel community describe this style of pull request evaluation as "no, by default." In this community, the contributions are often either not thoroughly evaluated or rejected without due diligence. Interviewee 24 stated, *"I communicate with the maintainer a lot. In general, he says no, unless he cannot say no. You know that is kind of his philosophy. I saw this view elsewhere in the Linux community"*. Winning the approval of the gatekeeper is critical. It requires persistence and accumulated trust (reputation). Interviewee 20 said, *"It's easy for me to get patches in because people in this community trust me and know who I am. Basic patches just go in easily because the maintainer trusts me. He knows that I will be around. If I submit a big chunk of code, and he does not know me, I may just disappear. Maintainers are very conscious about whether I know this guy ... the maintainer has to trust that the person will be around"*.

This attitude appears to be a gate that signals specific beliefs, such as the fact that commitment to the community must be demonstrated by the potential contributor, and winning the approval of the gatekeeper is critical. This necessary trust between contributor and gatekeeper comes from an ongoing relationship between the two individuals that exhibits trustworthiness. Once the contributor succeeds in dealing with the "no", then, the contribution is evaluated for its technical merits as explained by this interviewee, *"On some parts of the kernel building trust is essential, and there is a clear social entry barrier. It has some downsides for beginners. Yet it's understandable, as changes in the kernel always come with some kind of maintenance overhead, and maintainers want people that have proven to take ownership of their contributions ... However, once a patch is considered, then it goes through thorough vetting."* (interviewee 24)

> ***Protective*** is the PR governance style that relies on trust, relationship building and the contributor's reliability.

The interviews data shows that the Linux Kernel community PRs evaluation process exhibits the characteristics of a protective style. It appears that the protective style of governance is distinctive to the Linux kernel community that prioritize trust, reliability, and the contributor-maintainer relationship.

4.1.2 Equitable. The equitable governance style is about being fair and impartial regardless of who is the contributor. It is transactional in nature. The PRs submission evaluation is concerned with technicalities and less with social aspects. Interviewee 3 stated, *"We try to be very impartial, we try not to make interactions very personal because code change isn't about friends it's not about being friendly it's about managing a technology. And so there is a very straightforward mechanism of submitting code changes"*. This was echoed by many interviewees in several communities. Another interviewee stated, *"It's very transactional, and that's just one way of doing it and it's a way that we like because it keeps personalities out of it and it makes rejections not personal ... Yes we tend to keep personalities to minimum"* (Interviewee 9).

In this style, the community principles overrule any leniency toward contributors. Rejection is not loosely applied, but it is a social responsibility. Interviewee 7 stated, *"Rejections of pull requests are a social responsibility and are taken with a fairness in mind"*. The community exhibiting an equitable style applies a set of principles seriously during the evaluation process. Interviewee 8 stated, *"There are principles for evaluating pull requests, and we religiously obey them...and we will usually reject a pull request if it doesn't hold up to these principles"*. FOSSASIA and Odoo communities appear to be equitable.

> ***Equitable*** is the PR governance styles that values fairness and rigorous application of community principles.

4.1.3 Lenient. The lenient style of pull request reviews is a tolerant and compassionate style of governance prioritizing growth and openness of the community. The lenient governance style was prominent in the data collected from the Coala community. Interviewee 27 explained, *"We accept errors. Instead of rejection, we embrace the enthusiasm of the contribution"*. *"My first PR was reviewed 65 times but not rejected"* (Interviewee 26). The community is willing to invest in the contributors abilities by mentoring them to learn how to submit PRs that meet the community standards. This investment has to pay off at one stage, as this interviewee explains,

Category	Code or Theme	Definition	Example of verbatim
Software Engineering Principles	**Quality**	Quality is a subjective concept to FOSS contributors. This subjectivity is offsetted by reaching a consensus about when a piece of code make a "quality" contribution.	*I am not sure there is a specific way to assess quality. We can read through the code and we know good code from bad code. It is quite subjective. However, in our community, there is a requirement for a minimum 3 reviewers to approve code. That makes it objective.* Interviewee 28
	Avoid Technical Debt	Technical debt is the owing inherited from a contribution when it doesn't meet certain quality and design requirements.	*I will not add something that increases my maintenance burden unless it's very compelling functionality or an obvious bugfix. I can't maintain a system I don't fully understand, so I like keeping things lighter and cutting off edge cases rather than adding technical debt I don't have time to pay off.* Interviewee 8.
Social Norms	**Trust**	Trust is the unyielding belief that the person is truthful and reliable.	*There are obviously criteria that have to do with the contributor, I would mainly look for reliability and trustworthiness of the contributor.* Interviewee 9.
	Mentoring	Mentoring is establishing a support relationship between a mentor and a newcomer. A mentor is someone who partners with a newcomer during his or her early period of engagement with the community. The mentor offers advice and guidance to help foster and promote the development of a newcomer. The mentor knows the community, its products and processes, and can be an effective source of advice and encouragement.	*I had a mentor for 3 years. He helped me to become a better developer and an effective member of the community* Interviewee 27
Product Vision	**Feature within the community vision for the product**	Some FOSS communities set a vision for their products. Contributions have to fit within the defined vision and goals.	*My pull request have been rejected because generally, the maintainer does not find the feature aligned with the goals of the project.* Interviewee 16
PR Governance	**Protective**	Protective means designed or intended to guard or shield the code base from undesired and low quality contributions. It operates based on trust, relationship building and the contributor's reliability.	*It's easy for me to get patches in because people in this community trust me and know who I am.* Interviewee 20
	Equitable	Equitable means fair and impartial, all contributions are judged for their technical merits and suitability for the community product's vision.	*Contributions are assessed fairly and based on their quality not the contributor. Sometimes, it feels transactional and unsupportive.* Interviewee 9
	Lenient	Lenient means tolerant for errors but at the same time it does not compromise quality. Contributors are mentored to elevate the quality of their contribution to mergeable standards.	*When I joined the community, my pull requests were not rejected. Instead, I was shown by the mentor how to improve them and make them mergeable. Now, I produce high quality code, because I learned.* Interviewee 27

Table 4: Examples of Categories and Themes with definitions

Communities	Protective	Equitable	Lenient
FOSSASIA		✓	
Odoo		✓	
DuckDuckGo			✓
Linux Kernel	✓		
Coala			✓

Table 5: The studied communities PR governance styles

"You can't spoon-feed the developers all the time either. They have to demonstrate their abilities" (Interviewee 27).

The lenient governance is based on the belief that any contribution is an asset that should not be ignored. A contribution carries an enthusiasm that should be leveraged for the benefit of the community. Interviewee 26 stated, *"We have a rule in our community that we never, ever reject a PR. Instead, we manage the contribution and improve it. We make every PR mergeable"*. Another said, *"Rejections kill motivation and, it is a rude thing. We instead steer the contribution to a positive direction by making it better, and get it merged"* (Interviewee 18). However, this is not a compromise of quality. Lenient communities ensure quality by mentoring contributors to elevate their contributions to mergeable standards. We observed that the DuckDuckGo and Coala communities appear to be lenient.

> ***Lenient*** is the PR governance style that reduces social barriers and assumes that every contribution can be elevated to a mergeable state.

The literature suggests that socio-technical factors interfere with these perceived strategic styles of governance. FOSS reviewers review social signals more than they reported they did [12]. Ford, et al. report that, while reviewers reviewed code most (64%), they also reviewed technical (28%) and social signals (17%). Even when they do not realize it, reviewers consider social signals. Developers

should stay aware of their image on various social networks. Sharing one's image on social networks makes one trust that a person is who they say they are. If one feels unsure about revealing their own identity, they should use a pseudonym frequently enough to make it recognized. Completing the online profile is also important. In summary, identity is very important in today's open source communities [12].

The protective and lenient PR governance styles show that in some instances, the person and the code matter, while, the equitable style focuses on the code quality. Interviewee 6 stated *"the quality is more important than the person"*. The commonality across these PR governance styles is safeguarding quality. Interviewee 25 explains, *"the willingness to insist on quality is key to the success of PRs processes in FOSS projects". "The process in place seeks the best. The best code quality possible"* (Interviewee 20). PR governance delivers good outcomes. It is achieved by both the creation and use of systems that ensure consistency and repeatability of processes.

Each PR governance reduces the threat of poor code. Consistency and governance create a culture of excellence. *"The quality is the main driver that drive our decision to either accept or reject a PR. The processes are there to support and control the decision-making"* (Interviewee 2). This perspective contribute to the sustainability of software quality in FOSS. Interviewee 2 further explains, *"first reliability of the code. Open source is ever changing, people come and go. High quality code and the ability to read the code and understand it is critical"*. This belief was echoed across the various PR governance styles. A voice from a lenient community said, *"We keep contribution's code quality in the check, but at the same time we are trying to be lenient towards contributors to really help them out to get the codes to the level where it can be merged"* (Interviewee 29). These beliefs and behavior create a sustainable culture for quality.

4.2 RQ2: Principles of Evaluating Pull Requests

4.2.1 Software Engineering Principles. In the three styles of governance, once the PR is considered, it goes through an evaluation against a set of software engineering principles. The proposed change must also add clear value to the project. As this interviewee explained, *"We measure the success of a pull request by its ability to add value to the application or the community. It could be for example a legitimate feature, a payment of technical debts, etc."* (Interviewee 27).

There is a strong belief among the studied communities that quality is supreme, and quality is seen as a necessary quality of pull requests. Interviewee 15 stated, *"In open source projects, we like to achieve higher code quality because it is open source and we will need to get good quality code"*. Another one asked how he evaluates PRs, he replied, *"quality, quality, quality ... it always comes first"* (Interviewee 3). Interviewee 20 went so far as to claim that *"there are people who give up; not everybody can write the required quality of code."*

In the studied communities, quality is constructed of seven principles: (1) PR atomicity, (2) maintainability, (3) avoiding technical debt, (4) passing peer code review, (5) Compliance with best practices, (6) documentation, and (7) passing tests. These principles are not always documented and communicated. However, reviewers are aware of them and claim to rigorously apply them. Interviewee 8 stated, *"we have a well-established set of principles by which we evaluate PRs and we say 'no' when a PR doesn't meet our standards"* (Interviewee 8).

PR Atomicity. Atomicity is a requirement that the PR should be composed from relatively independent parts that can be understood separately and (possibly) reused. Our interviewees were aware of the evaluation criteria for PR atomicity, and they made it clear that atomicity is a key aspect of quality. Interviewee 27 stated, *"a pull request should be addressing one atomic concern and not more"*. The concept of atomicity is a common belief. Interviewee 9 stated, *"messy and bulky code is no good in open source"*. That it seems atomicity is ingrained in FOSS contributors' behavior. *"Anything more than 50 lines of changes, and my brain doesn't have the capacity to do a good code review"* (Interviewee 8).

Maintainability. Coleman, et al. [5] define maintainability as *"the ease with which a software system of component can be modified to correct faults, improve performance or other attribute, or adapt to a change environment"*. The interviewees strive to achieve code maintainability. Interviewee 8 states *"this is open source, we have to keep maintainability in mind all the time. The code must be neat and tidy and caters for long term changes"*. Maintainability also includes looking after the long-term of the project. Interviewee 24 states *"so many projects get derailed by accepting too many new features without evaluating them for long-term maintainability, and it is a problem that is avoided by a simple two-letter word - no."*

Technical Debt. The term "technical debt" describes a universal problem that software engineers face, which is the problem of how to balance immediate value with long-term quality. The term refers to a shortcut made for expediency, bad code, or inadequate code. This "debt" accumulates and causes increasing costs, or interest, to system quality in maintenance and evolution. This debt can be taken on deliberately, and then monitored and managed as principal repaid in order to achieve business value. Architectural choices are the major source of technical debt and often occur as a result of emphasis on fast delivery of features and limited budget [11].

Some interviewees in this study indicate an awareness of technical debt and its effects. They actively look at avoiding it. *"I will not accept something that increases my maintenance burden ... I can't maintain a system that I don't fully understand, so I like keeping things lighter and cutting edge. I strive to avoid technical debt, which we do not have time to pay off"* (Interviewee 17). Our data shows that maintainability and avoiding technical debts are tightly connected. Avoiding technical debts enhances maintainability and assuring maintainability encompass avoiding technical debt. Technical debt is a contingent liability with impact on the internal software qualities, primarily, maintainability and the evolution of the software.

Peer review. Before a code contribution can be added to the code repository, it must receive a positive review by a pre-determined number of reviewers, usually three to five. *"We have a definite principle that we have five reviewers that must approve the pull request"* (Interviewee 4). Each reviewer examines the code visually and subjectively to assess its quality. Reviewers provide necessary feedback concerning the code review. If they submitted code that does not meet the reviewers' judgment of quality code, then the code goes through cycles of iterative improvements until it is deemed good

enough for the code repository [1]. The studied FOSS communities believe that peer review is the mechanism that assures quality, that it is a valuable quality assurance practice. Peer code review is religiously adopted in the studied FOSS communities. *"There is no PR assessment without code review obviously. We have this non-negotiable rule that every PR must pass code review"* (Interviewee 1).

Best practices. The studied communities have agreed on best practices for the programming languages they use. During the evaluation of PRs, reviewers make sure that these best practices are applied. *"Pull requests reviews must follow the community best practices"* (Interviewee 4). In some communities, best practices go beyond the coding conventions and guidelines. For example, in FOSSASIA, the contributors' conduct is also covered with a best practice evaluation. *"First thing is when we sign up for FOSSASIA contributing, there is a list of rules that we have to follow, and these include being nice to people who are around you, and secondly is the code and standards for the code. The next thing is that we do not merge anything and everything that comes to the repositories"* (Interviewee 5).

Documentation. In FOSS, the documentation usually explains how the code operates, how decisions are made during the programming, and how to use and amend the code. *"We really focus on documentation because we believe a project can strive in a community with knowledge being documented"* (Interviewee 3).

Tests. The communities that we studied use various types of testing, such as unit testing, continuous integration, and integration testing. During pull request reviews, reviewers look to see if the PR has passed the necessary tests. *"We make sure there are proper tests to verify that a pull request works as expected. Pull requests will not be accepted without the proper tests"* (Interviewee 27).

> Once a PR is considered for a review, a set of software engineering principles are applied to assess its eligibility to be merged.

4.2.2 Social Norms. "Norms are properties of a group, they describe the typical or desirable behavior of a certain social group." Individuals know what behaviors are expected of them because social norms are communicated through verbal messages and modeled behaviors. Those not abiding by social norms are identified informally by social cues such as being isolated or rejected. Social norms are powerful and effective, and they are less resource intensive than incentive based or punishment systems [26]. We identified three social norms:

Trust. Trust is defined as the willingness of the community to rely on the contributor, the principle of trusting the contributor as a precondition for considering his or her code change. We observed this in the Linux Kernel community. This principle is unique to the protective governance style. Interviewee 24 stated, *"Changes to the kernel can be complex! I need to be able to trust the contributor to the point that I know he will be around to take ownership of the code".*

Establishing trust requires time. This time element makes it an entry barrier for newcomers. Other communities seem to have addressed this type of barrier and aligned the process to work solely with community principles. Interviewee 30 stated, *"We don't have entry barriers, but we ask the newcomers to obey our principles".*

Contributor-Maintainer Relationship. Having a relationship with the maintainer is an advantage in the process of getting a pull request accepted. Interviewee 21 stated, *"What helped is that I meet these people in person. It's a basic human thing. When you meet a person, it's not like a mailing list. Actually, it's a physical thing; you release a chemical called oxytocin".*

Mentoring. Mentoring is a practice put in place by some communities to help less experienced contributors to meet the community standards. Experienced contributors and evaluators take the time to work with the contributor to improve her submission. This action encourages additional submissions by that person and other observers as well.

Mentoring was observed in FOSSASIA, Coala, and DuckDuckGo among others. Interviewee 13 described mentoring, *"Pull requests that cannot be merged require mentoring. We have enough patience to work with the contributor to get it into a mergeable state. We mentor the contributor to do so"* (Interviewee 13).

> Trust, the contributor-maintainer relationship and mentoring are norms that take place during the evaluation process of PRs.

4.2.3 Product Vision. Some communities define a roadmap for their product and document it. During the evaluation of PRs, the proposed change is assessed whether it fits within the defined roadmap. *"We do not like to say no but we do to protect the evolution of the project"* (Interviewee 9).

> Pull requests proposed changes must adhere to the community roadmap for its products, in order to increase their chances of being accepted.

4.3 Trustworthiness and Limitations

Qualitative researchers pursuit trustworthiness for validity and credibility [31]. Trustworthiness is ensured by the establishment of these four traits: credibility, transferability, confirmability and dependability. Credibility refers to the confidence that the qualitative researcher includes the truth in the research study's findings. Transferability is the quality of the research demonstrating how the qualitative research can be applied to other contexts, that is similar situations, similar populations, and similar phenomena. Confirmability refers to how neutral the findings of the research study are, or how true the premise is that the responses are neutral and do not show any potential bias or personal motivations of the researcher. Dependability refers to the assurance that the study could be repeated by other researchers and that the findings would be consistent [18].

To establish credibility we used peer debriefing and member checking. One author conducted the coding and the other two authors validated the emerging codes and categories against the raw data. Six debriefing sessions were organized. We also used members checks to enhance the validity. We used it for narrative accuracy checks, and interpretive validity. We sent the interviews transcripts and description of the findings to the participants for validation. We collected data from five communities. This should strengthen the transferability of the findings.

We also used an audit trail to document and track the decisions we made throughout the study. This allowed us to meet confirmability requirements. An audit trail is the details of the process of data collection, data analysis, and interpretation of the data. To ensure dependability we compiled a research method that is s logical, traceable, and clearly documented [38]. When the research process is described thoroughly, the research audience is in a better position to judge the dependability of the research. If the process of the research can be audited, then it can ensure dependability [18].

Limitations. Linux and FOSSASIA interviewees were recruited through our contacts in the community. This make the Linux and FOSSASIA participants sample convenient. Convenience sampling has its criticism; it may not be representative of the targeted population.

5 DISCUSSION

The pull request governance styles have a reason to exist. Some are legacy, a result of years of ingrained culture and practice. Some are well crafted strategies put in place after years of trial and error learning. In all cases, these governance styles impacts their respective communities.

5.1 Protective

The protective governance style may create a *"clique"* culture difficult to access for newcomers. Newcomers may feel less important than the established core members of the community. Yet the community always needs newcomers, as creativity requires fresh minds and an ongoing flow of ideas and new contributions. Ostracizing those who are not inside the community may hinder its evolution and sustainability.

However, the protective style remains a good fit in some circumstances, such as when the FOSS project requires tight control over its code. It may be the appropriate style for a community to choose when they have an ongoing project where an influx of newcomers is not important, or when it is high anyways. Tsay et al. write that well-established and mature projects are more conservative in accepting pull requests [39]. The Linux community is a successful and mature project. Berger and coauthors note that the Linux community is a "closed platform" using heavy-weight processes. They also describe it as being a "centralized" structure; patches have to pass thorough reviews through the maintainer hierarchy [3]. They find it a justified practice when the project is developing a highly technical system, with a high barrier of entry, and high risk of introducing critical problems.

The protective style assigns relatively higher importance to commitment, relationships, and trust. Dabbish et al. report that both the contributor and the community look for signals of commitment. Frequency of recent submissions and the volume of activity by developers is a useful signal to the maintainer, while historical activity allows potential contributors to infer how well the project was managed. Visible actions on artifacts indicate the intentions, competence, and experience of the developers. Community support is inferred from the attention given, such as following, watching, and comment activity [7, 8, 23].

Relationships were shown to influence the evaluation of PRs. A chance of acceptance is higher for submitters already known to the core members of a project [40]. Also, maintainers interact more politely in discussions with core members than with new submitters [40]. The social connections between members of each of these groups can be measured on social distance and prior interaction values. Strong social connections increase the likelihood of acceptance, as they are markers of trust and allow to lower the assessment and coordination costs [33, 39].

5.2 Equitable

Although an equitable style of governance focuses on fair assessment, it does remain quite rigid. It is not a suitable style for communities that want to grow fast and attract new contributors, especially those with limited programming experience. Yet, this is the most preferred style of governance among the respondents overall.

The equitable style is suitable for communities aiming to attract experienced developers who are able to understand and incorporate advanced software engineering principles into their contributions. However, it does entry barriers for newcomers. Steinmacher et al. identified a list of barriers for newcomers, amongst them the need for orientation and technical hurdles [35, 36]. Communities which opt for this style of governance should communicate their evaluation principles clearly to contributors. They should educate contributors about their software engineering principles in their documentation.

5.3 Lenient

Mentoring contributors is a key part of the lenient governance style. This style is particularly well suited to communities with contributors with varied but limited experience in software development. An acceptance of the first contribution is an important step in a newcomer's socialization. She or he can learn the conventions and contribution rules through observation, lurking, and direct mentoring from more experienced members. Successful socialization allows potential contributors to learn the project norms and to identify the core members, where newcomers need to recruit allies [40]. After an initial period of observation, lurking, newcomers can assimilate the norms and values of the community. Then they begin to build an identity and become more visible to the core members, enrolling allies in the community. Once they demonstrate that they have the technical expertise, they are accepted by a community. Then they become an insider, not simply crafting material artifacts, but maintaining social relationships as well. They become a maintainer of the project, coaching and mentoring newcomers [10].

Attracting newcomers to communities is a major challenge. Fear of rejection that may harm reputation hinders some from contributing [16]. Lenient communities are aware of this issue and employ a strategy that minimizes rejections. Project members should show empathy toward new contributors, be engaged, and demonstrate fairness and positive attitude as mentors. Responsiveness and clear roadmap have also been identified by others as important factors encouraging newcomers [16, 17]. Berger et al. define variability encouragement as an open attitude to contributions from a broader ecosystem [3], and observed that some very fast growing ecosystems have openly and actively designed their processes and architectures to encourage external innovation.

Sim and Holt explain that a major downside of mentoring is that it is very time consuming for the senior developers in the community [32]. To some extent, the time required is compensated by attracting newcomers more easily.

5.4 Community Governance Vs. PR Governance

"Every development organization makes decisions and has some form of governance – this may be done explicitly or implicitly" [4]. FOSS projects are characterized by a specific framework through the lens of transactional cost economics called "bazaar" governance. This mode of governance is neither market nor hierarchy nor network, but is a governance system in its own right [9, 19].

FOSS governance is seen as the means of achieving the direction, control and coordination of autonomous individuals or organizations [22]. Community managed governance features are independence, pluralism, representation, decentralized decision making, and autonomous participation. The communities have a diverse group of participants that rests with the members of the community itself. An independent community allows decisions to be made at the lowest levels of the hierarchy, volunteers who may not be paid for their work. An independent community is deemed independent by its basis of material support, decision making structure, and independence from authority. A pluralistic community has a geographically diverse base of developers, community members who use a variety of ways to manage conflict, and leaders that emerge. Decision making occurs at the code level, the sub-project level, and the community wide level. Examining how members gain code level access rights, decision making rights, and the degree to which project communications and activities are publicly available lets one determine the mode of decision making [25].

PR governance is a process governance. Richardson [28] characterizes process governance as "consists of the set of guidelines and resources that an organization uses to facilitate collaboration and communication when it undertakes enterprise process initiatives." PR governance is the set of rules and controls that take place during the process of pull requests evaluations. It is doing what is required to assure that quality is produced by the process in the most efficient and effective manner possible. This is governance at the operational level of the community. Whether there is a link between community level governance and operational level governance is not something we explored.

6 RELATED WORK

The topic of PR-based collaboration has attracted some attention recently [16, 17, 21, 23, 27, 33, 34, 41–44]. To our best knowledge, no prior work attempts to conceptualize and distinguish the different governance styles in PR-based collaboration.

Soares and coauthors [33] find that the chance of a merge is 32% lower for first time contributions, supporting our intuition that the protective and equitable styles of governance are unfriendly to newcomers. In general, the chance of acceptance for a PR is 17% higher when tests are included, and 26.2% lower when many lines of code are changed [39]. This is inline with our findings, that passing tests and modularity of contributions are key criteria applied in evaluating PRs. The study has also shown that social distance and prior interaction with the maintainer are key influencers on acceptance chances [39]. This is consistent with our observation that social connections, trust, relationship building and commitment to the community, are considered in the PR evaluation processes.

Tsay et al. [40] note that maintainers were particularly concerned with the appropriateness of the contribution's actual content and direction. Appropriateness in this study is defined as fitting the product vision set by the community. We concur that adhering to the product vision is one of the evaluation criteria for PRs in the studied communities.

Marlow, et al. [23] study examines how interpersonal impressions influence evaluations of others' contributions. The analysis identified three scenarios where users sought out more information about each other. These scenarios are discovery, informing interaction, and skill assessment. Individuals form impressions about specific areas of expertise so that they can assess ways the coder can assist the project. They also make judgments about individual's personality. Arguments or rudeness in posting often are seen as indicators of uncooperativeness or arrogance [23]. This study concurs and complement our findings. It confirms that social inferences are part of the PR evaluation process. It complements our findings by suggesting that GitHub social signals are leveraged to make social inferences about contributors.

In FOSS communities, proper evaluation is seen as more important than addition of a feature. Developers prefer to postpone reviews rather than rush through them [29]. We observed similar attitude amongst our interviewees. They prefer investing great care and attention to detail rather than following a pre-defined checklist. This rigour coupled with the passion for the project lead to excellence in the evaluation process.

7 CONCLUSION

Modern software engineering heavily relies on open-source software. FOSS communities mostly emerges and organizes organically [37]. Measuring and tracking the organizational structure type and characteristics of an observable community is critical to achieve quality because identification of these systems provide organization problems that recur, such as motivation or trust, isomorphism, software failures, lack of centralized management of leadership, and stagnation. Software engineering research still lacks reference quality and models, but measuring community quality models can improve quality of software [37].

The PR governance styles foster a productive development and ensures high code quality. The controls and rules aim to improve the quality of source code changes made by developers, and it is a transparent process.

The PR evaluation process has a significant impact on contributor's motivation, so it is important to understand it. There is much more to learn from contribution evaluation than just simply whether the contribution is accepted or rejected, for instance we can understand better how to behave as community managers, reviewers, members, and how to enter communities more effectively. PR governance styles can be protective, with tight control of contributions; equitable, with a focus on technical fairness; and lenient, prioritizing community growth and retention by means of mentorship. Clearly, software engineering principles are not the

only criteria applied in PR evaluation; social and strategic criteria are also of high importance.

ACKNOWLEDGMENTS

Work supported by the XYZ programme, grant No 999999. We thank the interviewees for making this research possible.

REFERENCES

[1] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wasowski. 2019. Why does code review work for open source software communities?. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 1073–1083.

[2] Maria Antikainen, Timo Aaltonen, and Jaani Väisänen. 2007. The role of trust in OSS communities—case Linux Kernel community. In *IFIP International Conference on Open Source Systems*. Springer, 223–228.

[3] Thorsten Berger, Rolf-Helge Pfeiffer, Reinhard Tartler, Steffen Dienst, Krzysztof Czarnecki, Andrzej Wąsowski, and Steven She. 2014. Variability mechanisms in software ecosystems. *Information and Software Technology* 56, 11 (2014), 1520–1535.

[4] Sunita Chulani, Clay Williams, and Avi Yaeli. 2008. Software development governance and its concerns. In *Proceedings of the 1st international workshop on Software development governance*. ACM, 3–6.

[5] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. 1994. Using metrics to evaluate software system maintainability. *Computer* 27, 8 (1994), 44–49.

[6] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 275–284.

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 1277–1286.

[8] Paul B De Laat. 2010. How can contributors to open-source communities be trusted? On the assumption, inference, and substitution of trust. *Ethics and information technology* 12, 4 (2010), 327–341.

[9] Benoit Demil and Xavier Lecocq. 2006. Neither market nor hierarchy nor network: The emergence of bazaar governance. *Organization studies* 27, 10 (2006), 1447–1466.

[10] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368.

[11] Neil A Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L Nord, and Ian Gorton. 2015. Measure it? manage it? ignore it? software practitioners and technical debt. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 50–60.

[12] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. 2019. Beyond the code itself: how programmers really look at pull requests. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*. IEEE Press, 51–60.

[13] Linux Foundation. [n. d.]. 2017 Linux Kernel Report Highlights Developers' Roles and Accelerating Pace of Change. https://www.linuxfoundation.org/blog/2017/10/2017-linux-kernel-report-highlights-developers-roles-accelerating-pace-change/

[14] Graham R Gibbs. 2007. Thematic coding and categorizing. *Analyzing qualitative data* 703 (2007), 38–56.

[15] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 345–355.

[16] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 285–296.

[17] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 358–368.

[18] E. G. Guba and Y. S. Lincoln. 1985. Naturalistic inquiry (Vol. 75). *Beverly Hills, CA: Sage* (1985).

[19] Kieran Healy and Alan Schussman. 2003. *The ecology of open-source software development*. Technical Report. Technical report, University of Arizona, USA.

[20] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research policy* 32, 7 (2003).

[21] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. 2017. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology* 84 (2017), 48–62.

[22] M. Lynne Markus. 2007. The governance of free/open source software projects: monolithic, multidimensional, or configurational? *Journal of Management & Governance* 11, 2 (2007).

[23] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 117–128.

[24] Matthew B Miles, A Michael Huberman, and Johnny Saldana. 2014. *Qualitative data analysis: A methods sourcebook. 3rd.* Thousand Oaks, CA: Sage.

[25] Siobhán O'Mahony. 2007. The governance of open source initiatives: what does it mean to be community managed? *Journal of Management & Governance* 11, 2 (2007), 139–150.

[26] Elizabeth Levy Paluck and Laurie Ball. 2010. Social Norms Marketing to Reduce Gender Based Violence. *IRC Policy Briefcase* (2010).

[27] Mohammad Masudur Rahman and Chanchal K Roy. 2014. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 364–367.

[28] Clay Richardson. 2006. Process governance best practices: Building a BPM center of excellence. *Business Process Trends* (2006).

[29] Peter C Rigby and Margaret-Anne Storey. 2011. Understanding broadcast based peer review on open source software projects. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 541–550.

[30] Colin Robson and Kieran McCartan. 2016. *Real world research*. John Wiley & Sons.

[31] Andrew K. Shenton. 2004. Strategies for ensuring trustworthiness in qualitative research projects. *Education for information* 22, 2 (2004).

[32] Susan Elliott Sim and Richard C Holt. 1998. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In *Proceedings of the 20th international conference on Software engineering*. IEEE, 361–370.

[33] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 1541–1546.

[34] Daricélio Moreira Soares, Manoel L de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 960–965.

[35] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*. ACM, 1379–1392.

[36] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. 2018. Almost there: A study on quasi-contributors in open-source software projects. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 256–266.

[37] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. 2019. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering* 24, 3 (2019), 1369–1417.

[38] Sarah J Tracy. 2010. Qualitative quality: Eight "big-tent" criteria for excellent qualitative research. *Qualitative inquiry* 16, 10 (2010), 837–851.

[39] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. ACM, 356–366.

[40] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 144–154.

[41] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for it: determinants of pull request evaluation latency on GitHub. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 367–371.

[42] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1. IEEE, 335–342.

[43] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[44] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 871–882.

F

Appendix F: Paper F

The 7 Habits of Good Pull Request Evaluation

Adam Alami
IT University of Copenhagen

Marisa Leavitt Cohn
IT University of Copenhagen

Andrzej Wąsowski
IT University of Copenhagen

ABSTRACT

There are over two million organizations actively using Github. This number grew by 40% from last year. This year, more than 96 million of the world's software projects collaborate in Github. This collaboration is managed by a process known as pull request (PR) where a contribution is proposed and discussed. PRs are evaluated by the receiving community for appropriateness and correctness before a decision can be made whether to merge or not. It may sound that the process is straightforward, but it's not—It is loaded with social norms and technical judgments. In this study, we aim at understanding contributors' experience during the course of PR evaluations. We collected 58 cases of fairly and unfairly assessed PRs. We find that a small number of good practices (habits) result in a positive experience for the contributor. The obtained framework suggests a set of good practices to enhance the PR evaluation experience for all stakeholders. It aims to help communities to improve guidelines for PR evaluation.

CCS CONCEPTS

• **Software and its engineering → Open source model**; • **Human-centered computing** → Collaborative and social computing.

KEYWORDS

Open source software, pull request, Code review

ACM Reference Format:
Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2020. The 7 Habits of Good Pull Request Evaluation. In ,. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Over two hundreds million pull requests have been submitted to GitHub, the third of this number only in 2018. These pull requests are the product of the collaboration of over 31 million developers.[1] A strong testimony for the growth and the popularity of pull-based development model. Pull-based development is a model of software development where the project's main repository is not shared among potential contributors, but contributors fork the repository and make their changes independently. The role of the integrator who merges the forks is a critical. The integrator acts as a guardian for quality while keeping many contributors actively involved in the community. She must facilitate consensus and timely evaluation of the contributions, while enforcing online discussion etiquette and on-boarding of new contributors. The two key factors that integrators are concerned with are quality and prioritization [11]. The most frequent challenges encountered by contributors to a project are social in nature, specifically lack of responsiveness from integrators and a lack of empathy from integrators, along with difficulties in communicating change rationale [10].

The evaluation process can have an important impact on contributor motivation, so it is important to understand the factors affecting evaluation [35]. The outcome of an evaluation is farther reaching than just whether the contribution is accepted. It impacts the motivation of a new contributor to contribute again. Only 15.38% of newcomers continue to contribute to FOSS projects [32]. The acceptance rate of contributions to FOSS projects is ca. 31%-40% depending on the community [2, 16, 20, 31, 36]. Lee et al. cite dissatisfaction with the process of submission as key reason why one time contributors abandon communities after the first accepted contribution [17]. The picture of PR evaluation found in FOSS studies is not rose. Most contributors see the integration as a process with an unpredictable outcome [16]. The objective of this study is to identify the contributors' experience when they submit a PR. To understand this we pose the following research question:

RQ: *What do contributors experience when submitting a pull request?*

To answer this question, we asked contributors to show us examples of pull requests evaluated fairly and unfairly (in their personal opinion), and to argue why they have selected those PRs. We performed a qualitative analysis of the comments and the PR discussions. Seven qualities emerged as determinants of good contributor's experience in pull requests evaluation.

We identified a set of seven habits (tongue in cheek)—good practices for pull request evaluations. When these practices are integrated into a PR evaluation, the process is deemed fair by contributors. When they are missing or violated, the evaluation is considered unfair. For example, we identified engagement as a good practice. When a PR receives the adequate engagement level, the process is likely to be considered fair, but engagement is difficult to achieve as it requires intensity. Intensity is the number of community members on top of the maintainer who are involved in reviewing and commenting. When the number of comments and reviews exceeds two, the PR review process exhibits a vibe and energy, generates strong motivation for the change.

Another example of a good practice is communication. The communication tone adopted during the evaluation process is critical for the success of establishing engagement when reviewing pull requests. When the tone of the communication is positive and professional, it steers the evaluation process in a positive direction. In addition to engagement and communication, we found that appropriateness, simplicity, compliance, support, and honest decision support the evaluation process positively.

[1] https://octoverse.github.com/

ESEM'2020, Empirical Software Engineering and Measurement (ESEM)

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Good Practices	Contributor	Maintainer	Community
Engagement			✓
Communication	✓	✓	✓
Appropriateness	✓		
Simplicity	✓		
Compliance	✓		
Support			✓
Decision		✓	

Table 1: PR evaluation practices and involved stakeholders

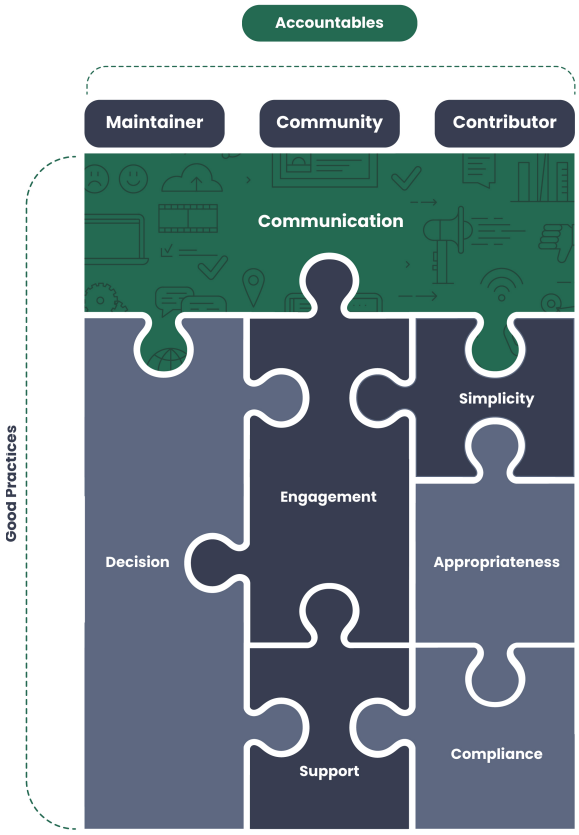

Figure 1: FOSS PR Evaluation Good Practices Framework

2 FINDINGS

For a contributor, submitting a PR is a journey with unexpected course of events and outcomes. Unknowns and barriers unfold when the stakeholders bring deficient practices into the process. Making the experience unpleasant and demotivating. Still, the journey can be smooth, with minimal breaks, no sudden changes and shifts, if stakeholders adhere to a small number of good practices.

Based on an analysis of past pull request discussions, we suggest a framework of good practices for PR evaluation to implement, document and communicate in FOSS communities. In this section, we present and discuss the framework (Fig. 1). The framework is stretched over two dimensions: (1) accountables and (2) good practices. Accountables are the stakeholders responsible for certain conducts. Good practices are the expected qualities and behaviours to take place during the PR evaluation process or conditions that the PR itself need to fulfill.

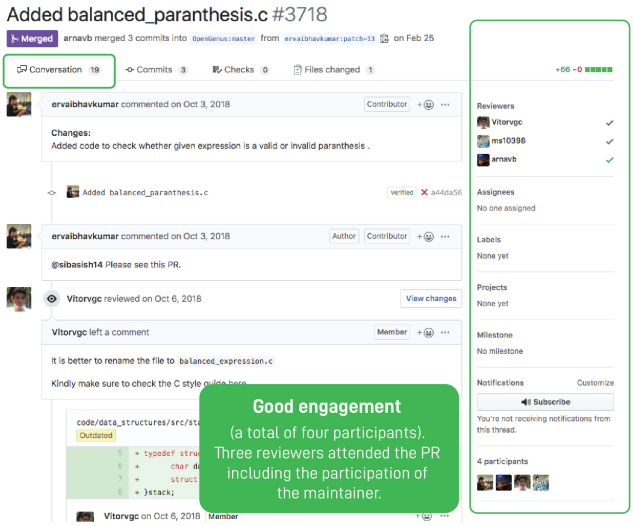

Figure 2: Example of a PR with good engagement

Our findings (Fig. 1) suggest that in order for contributors to enjoy their journey, the stakeholders of the PR evaluation process should habitualize several practices: (1) engagement, (2) communication, (3) appropriateness, (4) simplicity, (5) compliance, (6) support, and (7) honest decision making. These practices are the responsibility of either the contributor, the maintainer, or the community in general. Table 1 illustrates the distribution of responsibilities across the PR evaluation stakeholders.

We collected the data, that was the base of our analysis, through a survey. We asked respondents to supply two pull requests, one fairly and one unfairly assessed. We collected a total of 58 cases. We also asked the respondents to argue for their selection. The details of the methods are summarized in Sect. 3.

In this report, we borrow names from Lewis Carroll [5]. Let Alice be the contributor. The uncertainty that Alice is confronted with throughout her time in Wonderland is akin to the environment inhabited by the PR evaluation process. To an external onlooker, submitting a PR may appear rooted in cold and hard facts. However, just like Alice, the contributor will be tested by the complexity of the working environment and its decision process. The culture of FOSS communities may be predicated on the assumption of an overall collective togetherness, hinging upon shared goals and values. However, each individual member is susceptible to the erratic behavior, conflicts of values and norms.

We use Queen of Hearts to name the maintainer. The Queen of Hearts exemplifies a dominant, authoritarian who wields a great deal of power when it comes to the implementation of change. Our intent is not to portray communities maintainers as authoritarian individuals with poor communication skills, but to attribute power clearly. The Cheshire Cat represents the PR reviewers. Through her interaction with the Cheshire Cat, Alice gains valuable insight into the nature of pull request evaluation.

2.1 Engagement

Figure 2 shows an example of a PR evaluation where Alice received an enthusiastic and engaging reception from the community. The PR discussion became animated and lively.

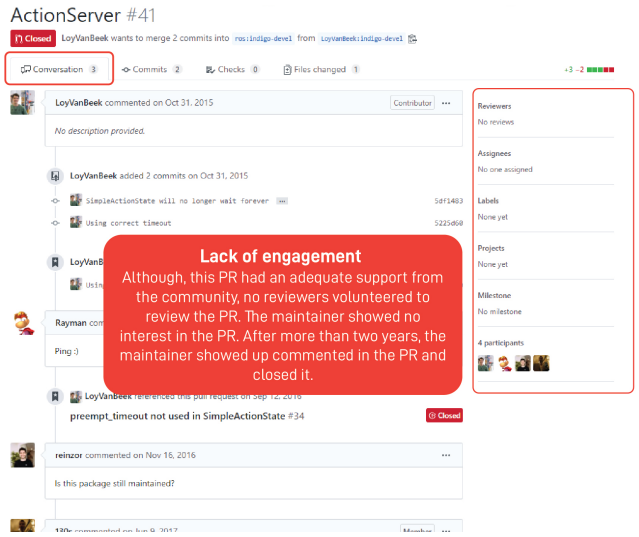

Figure 3: A pull request discussion lacking engagement

Engagement, designated as important by our respondents, is the involvement of the community with the contribution. A respondent writes, "The community showed interest in the PR. More than three reviewers and the maintainers reviewed the PR. This was really motivating." Engagement acknowledges the contributor's effort, and motivates her. Engagement is the level of participation from the community that a PR receives.

In the example of Fig. 3, Alice has experiences a rather cold and unengaged reception. This could be due to many reasons; Alice could be a newcomer to the community or her suggestion for change was not appealing to the community. She explains, "No one has shown interest in my PR. Worse! It was closed abruptly by the maintainer without explanation. I left this community because of this." The lack of engagement is demotivating, and being a very high costs: the loss of contributors, who are vital of the community.

Lack of engagement is demotivating. A respondent said: "It is the maintainer not engaging the community. The pull request has a fair number of people commenting that the change would be beneficial and more people giving thumbs up as feedback. There has been no maintainer feedback for more than two years. It is very unlikely that other contributors will attempt to provide updates to this repository in the future." Another respondent associated unfairness to lack of engagement, "most of the PR that I consider unfair are the good ones that get unnoticed." Various factors contribute to this behavior. It may be that the contributor does not have a relationship with members of the community, the PR's scope is not interesting or not a good fit for the community's needs. Regardless of the reason, the contribution should be acknowledged, and the reason why the community is not willing to invest time in the PR should be given to the contributor. Otherwise Alice experiences the "Mad Tea Party" where the attendees are oblivious to her presence.

Intensity. The participation of an appropriate number of community members is important to the contributor. One respondent commented, "This PR got input from several people. Friendly and open discussion focusing on technical points."

Intensity is the number or reviewers who interact with the PR. The right intensity brings momentum and energy to the PR evaluation process. It signals acknowledgement and appreciation for the value of the contribution and for the effort invested by the contributor. However, excessive intensity encourages divergence and delays of the finalization of a contribution. Intensity is also demonstrated by the depth of the discussions of the PR and a positive attitude toward the contribution. Sometimes, keeping the discussion technical is the sign of a motivated reviewer.

Enthusiasm shown by the reviewers and the maintainer for the change and its value give Alice a positive experience. Helping her to move the process forward. Enthusiasm can also be shown with other signals like emoticons.

> **Engagement.** Every contribution carries enthusiasm, which should be rewarded in the form of acknowledgement, feedback, and a decision. It is a good habit for maintainers and senior community members to ensure that new contributions are discussed timely, that feedback is provided, and decisions are made.

2.2 Communication

The Mad Tea Party [5] event completely contradicts the expected flow of a rational social situation. The erratic changes in conversation and unpredictable topics defy expectations and violate social norms. Abrupt changes of subject prevent a constructive conversation. Figures 4 and 5 show examples of a constructive and toxic communication respectively. A participant in Fig. 5 opens an abrupt verbal attack on the maintainer, which encouraged another participant to follow up with a profanity. These toxic comments have incited others to engage in a cascade of hostile and divergent comments. Although, the rejection of the PR was due to an ideological polarization in the community, the communication style adopted by the participants exacerbated the situation. In Fig. 4, however, the Cheshire Cat (i.e. one of the reviewers) showed enthusiasm and positive attitude toward the contribution, which helped to set a positive and constructive tone.

The style of communication adopted by the participants in the PR evaluation is critical to the success of the evaluation process. Alice (in Fig. 4) explains: "The tone of the conversation was constructive. In the end, improvements were made to the software." It is advisable to keep the discussion technical. Alice elaborates: "The comments were valid, and the discussion was technical." Maintaining a technical discussion throughout the evaluation process is seen as objective conduct. Respondents prefer a communication tone that is professional, constructive, and supportive.

> **Communication.** Maintainers and reviewers should develop a habit to communicate objectively, clearly and professionally.

2.3 Appropriateness

Contributed features must meet the needs and plans of the community. Appropriateness is explained in a comment by a respondent who states: "The changes suggested were necessary. We obviously, don't like unicorn features! We like to invest our energy in something that adds value. As soon as the code quality and correctness were good, it got merged and got appreciated." The adherence to the community's needs and standards, whether a required feature

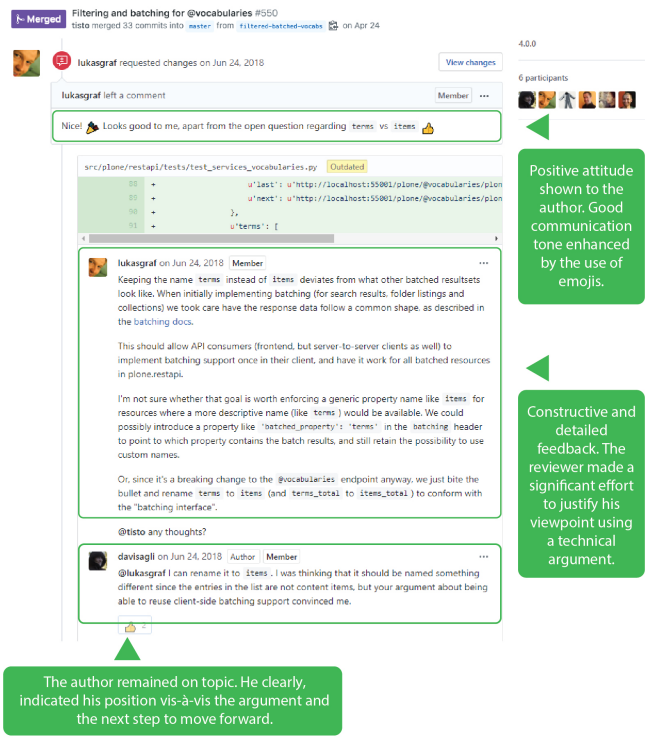

Figure 4: Example of a Constructive Communication

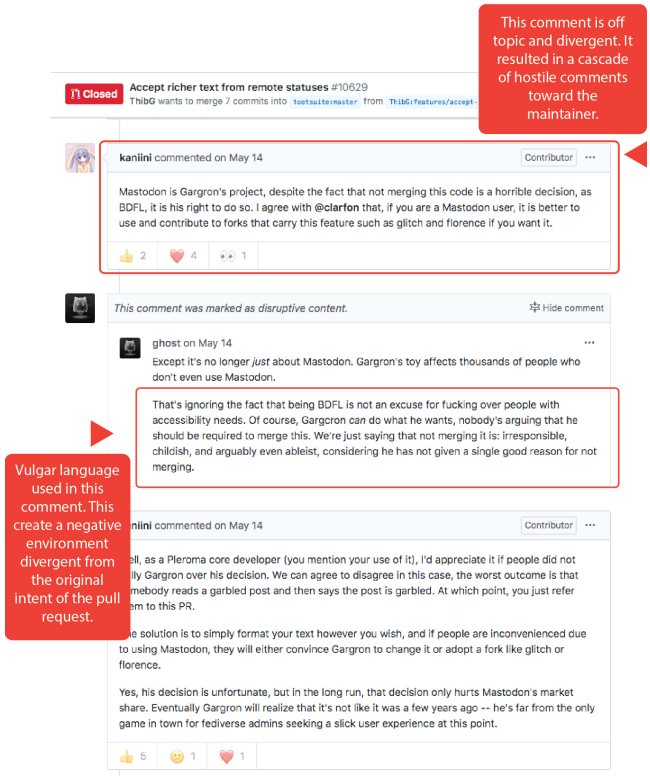

Figure 5: Example of a PR with Toxic Communication

within the scope of the product or fixing a bug, is valued in the process. A maintainer replied to the frustration of a contributor when his PR was not merged: "PR cannot be merged on your sentimental. Please respect the mentor's feedback. If your PR is not fulfilling the requirements, it cannot be merged, simple as that." This uncompromising position shows the significance of this practice.

> **Appropriateness.** Contributors should develop a habit to consult the community issues list and the product vision documentation when available, to ensure that their contribution addresses a legitimate need. Communities should document their product vision and promote it towards contributors.

2.4 Simplicity

Simplicity makes one feel in control, apt to judge and to act. It makes things elegant, obvious, and transparent. To succeed in FOSS communities, Alice should be able to translate complexity into credible familiar simplicity. When Alice represents things in understandable form, that is, in the universal language of common sense, she diminishes uncertainty.

This practice is well supported in the PRs we examined, and in the comments left by our respondents. Alice stated, "the history is messy because of the number of times it was rejected, but it was fairly rejected, which resulted in a more modular, better-tested code than the initial commit." Another respondent praised a PR for its simplicity. He stated, "This is a good example for a modular and straightforward PR." Another respondent explained that his PR was unfairly judged because of the risk it may induce to the code base. Alice explained "Reviewers didn't evaluate the PR based on the improvements it would make, but based on fear it would break things. IMHO we should've worked towards making the PR more robust, if the reviewers had concerns." Complicated changes are risky, and communities are not willing to take such risks.

FOSS communities prefer straightforward and modular changes. Modular changes are easy to review and their quality is easy to assess. The reviewers can efficiently understand the context and the reason of a straightforward contribution. Alice may need to spend more time breaking down the problem while solving it. Cumbersome pull requests also increase the risk of introducing defects since the changes are so big. It is hard to test large monolithic change sets for bugs and regressions.

> **Simplicity.** This quality is highly appreciated in the contributions. Contributors should develop a habit of developing and submitting easy to understand and modular submissions.

2.5 Compliance

FOSS communities like rules. Rules agreed upon are strictly enforced. Compliance is mainly the adherence to the community programming best practices and other non-programming guidelines. Most communities have implemented guides and best practices. The scope of these guides varies amongst communities. While some are limited to programming language guidelines, other are extended to cover conduct, communication and community culture. Following best practices enhances the chances of a PR being accepted. This respondent explains, "it was following good practises, was

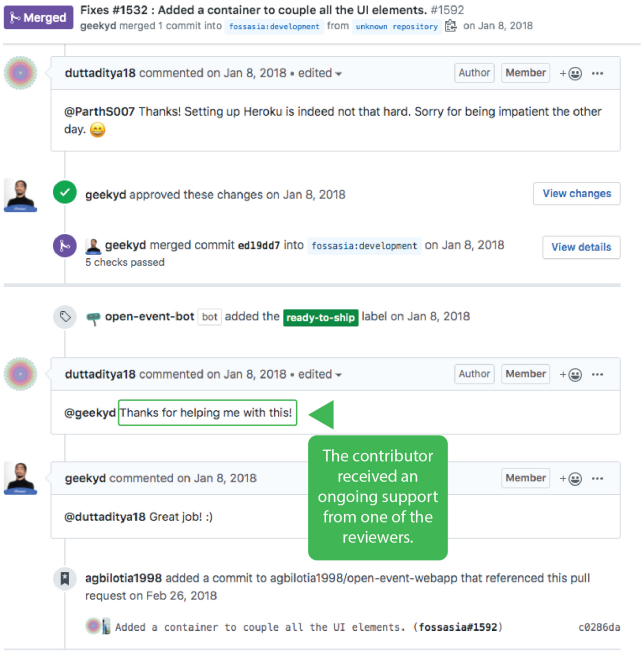

Figure 6: A pull request that has received support

reviewed by two contributors where one of them was main project maintainer. And it got merged."

> **Compliance.** Contributors should ensure that their submission is compliant with community guidelines and rules. Compliance signals competence of the contributor and recognition of the competence of the community.

2.6 Support

In FOSS communities, receiving support to improve a code change submission is not guaranteed. Figure 6 illustrates an example where Alice received support from the Cheshire Cat (i.e. reviewer). This support produced a productive outcome, a compliant and mergeable quality PR. Unfortunately, not all Cheshire Cats are supportive. Figure 7 demonstrates a transactional and cold encounter with the Cheshire Cats. Navigating through the unknowns for newcomers can be daunting. Alice asks for help, but receives brisk technical critique with no clear direction.

One respondent stated, "Even though I was unable to understand and follow the guidelines, the maintainers helped me ..." Another participant in the survey describes support with this comment, "the contributor was guided through steps to make the code better and more useful than what have been initially created. The code was reviewed and put to the test." Support does not only enhance team cohesion, but also help elevating the quality of the submission.

"No one helped me." This is how Alice defined an unfair PR evaluation. When we examined the PR history and course of events, we observed that the PR lacked engagement and support. Lack of support is characterized by minimalist feedback and focus on communicating the errors and not on how to fix them. The communication tone is usually rigid and lacks empathy toward the

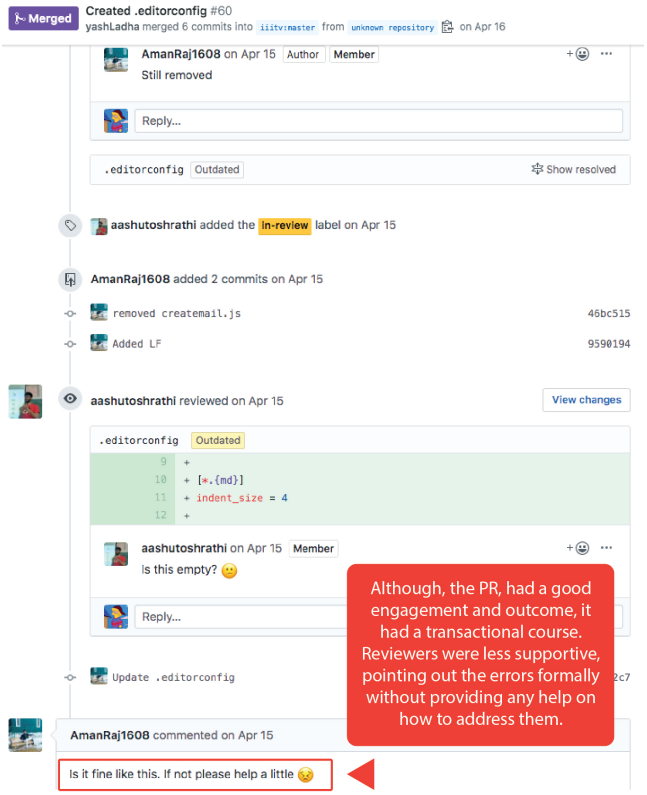

Figure 7: A pull request that has received little support

contributor. Respondents have clear preference for a supportive evaluation process and a constructive feedback that help steering the PR toward a mergeable quality.

> **Support.** A supportive evaluation process creates a positive atmosphere and gives a good experience to contributors. Reviewers and maintainers should develop a habit of being constructive and collegial.

2.7 Decision

Alice could not find meaning in Wonderland but hopes that she will find logic and order in the trial. She thought that Wonderland court is a true court of justice, viewing the institution of law as a refuge of sanity in which an objective and undeniable truth will prevail. However, the trial was a disappointment.

Unfair decisions are common experience in FOSS communities. We observed from our data that contributors expect a logical decision based on technical merits. Figure 9 demonstrates an example of unfair decisions. Alice, enthusiastically, raised a PR to address, an apparently legitimate issue. She was faced by a cold reception, no Cheshire Cat attended her PR for a year, when someone spoke, but haven't advanced the discussion. Alice has been repeatedly asked to sign the contributor license agreement (CLA), which she already did shortly after submitting the PR. The PR was closed automatically by a bot after one year and five months. Alice perceived the decision as unfair, and it does not seem to help that it was made automatically. She invested great effort in writing a PR of mergeable quality and

Figure 8: Example of a PR with a Fair Decision

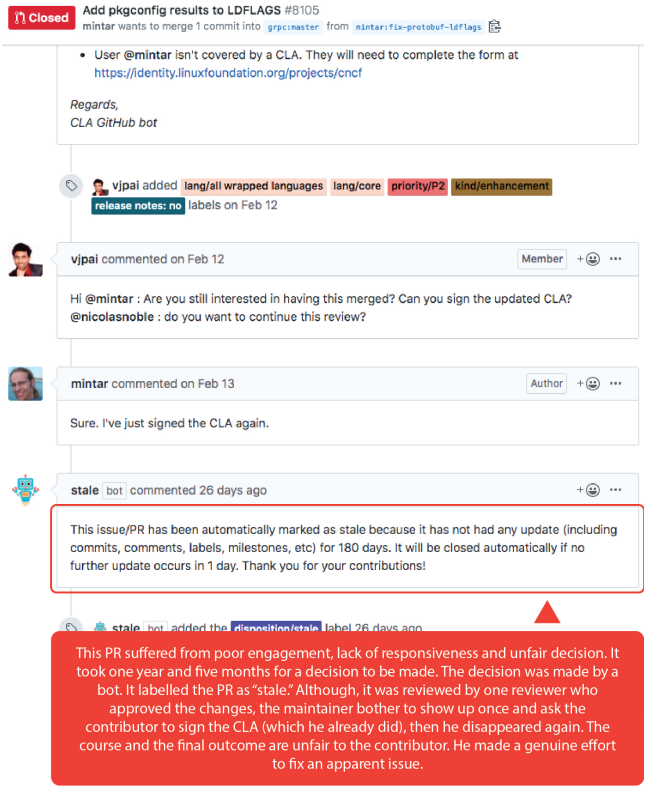

Figure 9: Example of a PR with an Unfair Decisions

praised by another reviewer for its appropriateness: "This patch fixes my current problem. Any chance this can be merged soon?" The decision, not to merge, was not based on technical grounds but rather a procedural failure.

Figure 8 shows an exemplary case of a PR that embraced good practices, which resulted in an orderly evaluation. Alice's PR received a great engagement from the community and the communication remained professional and within the scope of the PR. Once two Cheshire Cats approved the PR, the PR became a candidate for merger. The PR was merged accordingly.

One contributor describes an appropriate evaluation experience saying, "It was judged on technical merit, the contributor was guided and the PR was merged in the end." The belief that fair evaluations are focused on technical merit is common. The basis of a decision from the evaluators is important to contributors and the health of the community. The decision is the conclusion about the fate of the PR after consideration. Decisions grounded on technical rationales are perceived as good practice by our respondents.

Executed in the style of the Queen of Hearts: Thoughtless, baseless in decision, can be easily be interpreted as a personal rejection. It is not a pleasant experience for the contributor and it may have negative impact on the contributor retention.

> **Decision.** Decisions should be based on technical grounds. Maintainers should develop a habit of communicating the decision rationale clearly, in technical language.

3 METHODS

Subject Communities. We selected eight communities for this study, ROS, FOSSASIA, Coala, Plone, Apache Spark project, OpenSUSE, Linux Kernel, and OpenGenus. These are well established open, and growing communities. ROS is creating a flexible framework for robots, including a collection of tools, libraries, and conventions to simplify the task of creating robot technology across platforms. FOSSASIA develops, among others, SUSI.AI, an artificial intelligent application that provides functionality for personal assistance, Help Desks and Chatbots and EventYaY, an events management tool. Coala is building an open source linting tool for developers. Plone is an open source software content management system. Plone has a long list of high-profile users, including the U.S. Federal Bureau of Investigation, Brazilian Government, United Nations, City of Bern (Switzerland), New South Wales Government (Australia), and European Environment Agency. Apache Spark is a unified analytics engine for large-scale data processing, built by developers from over 300 companies. OpenSUSE builds open-source tools for software developers and system administrators, such as openQA, an automated testing service; OSEM, an event management tool; Jangouts, a videoconferencing tool; YaST, the installation and configuration tool; and Kiwi, an application for making a wide variety of image sets available for Linux supported hardware. The Linux Kernel is a free and open source operating system. Nowadays, likely the most popular and versatile operating system kernel. OpenGenus is a FOSS community whose goal is to enable longer off-line working conditions and to reduce the time spent on search. Its project Quark enables people to work offline for a longer stretch of time, and its search engine reduces the time spent on searching.

Data Collection. We collected the data through a survey where respondents were invited to participate based on emails from GitHub. Some participants were found on community forums and mailing lists. We asked the respondents to provide two cases of pull requests, one where they felt the pull request was assessed fairly, and a second one where they felt that it was evaluated unfairly. In both cases, we asked for justification of the choice. We received 48 cases of fairly assessed PRs and 10 unfairly evaluated PRs (we studied N=58 cases). Participation in the survey was voluntary, so no compensation in any form was given to the respondents.

Instruments. The survey consisted of a set of multiple-choice questions and questions that can be answered with open ended text. There were a total of twelve questions. This paper reports the findings of the data collected from the questions mentioned above.

Data Analysis. We conducted a thematic analysis of the explanations for the categorization of the decisions as fair or unfair and the PRs contents. The comments and the PRs contents were coded following the guidelines of Robson and McCartan [25] and of Miles et al. [19] concerning qualitative analysis. The text were examined line-by-line and coded. Once all the text were coded, we looked for patterns amongst the codes to propose themes that emerged from the data. These themes share meaning across data items and are underpinned by a central concept that answers a specific research question. Once labels are given to each of these concepts, then we were able to place them in categories, or a family of codes.

Trustworthiness. When performing qualitative research, researchers must establish that the findings are credible, transferable, dependable, and able to be confirmed. Once these four qualities are established, trustworthiness is achieved [12].

To establish credibility, we used data triangulation and peer debriefs [7]. Triangulation is the combination of at least two data sources. The aim of using triangulation is to decrease the deficiency of a single strategy, thereby increasing the ability to interpret the findings. We have two sources of data, the comments left by the respondents in the survey and the PR's discussions. To triangulate the data, we compared the findings across the data sources. We ensured that our findings are supported by the two data sources. We conducted a couple peer debriefing sessions. One author conducted the analysis, and the other authors validated the theory that is emerging with the raw data.

Transferability is the degree to which the results of a study can be transferred to other contexts, sites, or settings [12]. The quality of transferability refers to case-to-case transfer [34]. We thoroughly documented our research methods and the analysis process which would helps others to judge transferability [12].

To ensure dependability, information that is logical, traceable, and clearly documented is given. When the research process is described completely, readers are able to judge the dependability of the research by auditing the research [12]. We maintained an audit trail throughout the study to ensure the traceability of our work.

Confirmability is the characteristic of the match between the researcher's interpretations and findings and the data itself, which requires the researchers to explain how the conclusions and interpretations were made [34]. In addition, it is helpful for the researchers to compile an audit trail, specifically documentation that explains the rationale for the decisions and choices that were made including both theoretical and methodological issues. This audit trail enables other researchers to follow the decision trail and reach the same conclusions.

4 DISCUSSION

PR nonacceptance might result in demotivation of the contributor, potentially preventing further contributions [26, 31]. There is a high turnover rate in communities, recruitment and maintaining new contributors is critical for the success of a community and its projects [9, 15]. Many FOSS projects fail [6]. Coelho and Valente seek to find out why [6]. They suggest that successful projects follow maintenance practices, while failed products did not. The maintenance practices that set successful projects apart from failed ones were the issuing of templates, code of conduct, and PR template documentation, continuous integration, contributing guidelines, and licensing were relevant to the success of a project [6]. However, the PR evaluation good practices are not documented and communicated in the communities we studied. This is an important and critical process to the continuity of a community.

4.1 All Stakeholders

Communication is the only good practice shared across all stakeholders. How to communicate a message is as important as the message itself. In the context of PR discussion, a message has two attributes, the content and the delivery. Both have to show certain qualities. While the content has to remain within the topic and constructive, the delivery should remain professional and friendly. Still, the process is highly social. Communication can be derailed for various reasons. But, in this instance, the maintainer has to demonstrate leadership and steer the process back to a positive course.

Geographically distributed teams experience more miscommunications and misunderstandings, have more difficulty sharing information, providing feedback and face more challenges in attempting to develop and maintain a shared team identity [14]. Members of a team can be categorized as constructive, passive, and aggressive styles. The constructive style is a balanced concern for personal and group outcomes, cooperation, creativity, free exchange of information and respect for others. This style enables members to fulfill both needs for personal achievement and affiliation. The passive style places greater emphasis on fulfillment of affiliation goals, maintaining harmony, and limited information sharing, with maintaining harmony as a goal. The aggressive style places emphasis on personal achievement, with personal ambitions placed above concern for the group. The aggressive team member demonstrates competition, criticism, interruptions, and impatience [21].

Constructive groups produce solutions that are superior in quality to those produced by passive groups and superior in acceptance to either group. Passive styles produce solutions that are inferior, and aggressive individuals produce solutions not as high in quality [21]. Lack of communication or trust between patch writers and reviewers lead to the rejection of the PR [33].

Recommendation 1. *FOSS communities should document, communicate and implement guidelines for contributors on how to communicate constructively. Maintainers should enforce the guidelines and exercise leadership when communication failure occurs.*

4.2 Contributor

Simplicity. The principle benefit of simplicity is that it provides conditions that are more conducive to the acceptance of the PR. Tao, et al. suggest that when the patch is difficult to read or maintain or the size is too large, then it is rejected [33]. Baysal, et al. report that smaller patches are more likely to be accepted and accepted more quickly [3].

Rigby et al. [23] found that small, independent, complete patches are most successful in the Apache server community, whereas, Jiang, et al. [16] found that size only influences the reviewing time. Rigby and Storey suggest that the file change should be short so developers can instantly see if they have the time, skill, and interest to perform a detailed review and if the patch is of sufficient quality to warrant a review [24]. More than half of all patches in FLAC change only one or two lines and small patches get accepted more frequently than average, at 57%. Large patches, more than 15 lines of code, are less likely to be accepted, with only 3.6% being accepted [36].

However, this is a problem for newcomers. Newcomers often lack confidence choosing the first task. Often they are not sure about what is easy or not. They do not know how to reduce the scope of a task. Documentation issues are part of the difficulty, specifically lack of documentation and incomplete or outdated tutorials. Another project problem is the information provided in the issues, there are outdated issues, lack of information about required skills, and lack of information about difficulty level. Sometimes newcomers choose a wrong or large task [29].

> ***Recommendation 2.*** *Maintainers should guide the contributors how to break down large changes into sequences of smaller ones. Thus, uncertainty is reduced. Simplicity can be implemented into the change by focusing on a few, but decisive must-win battles. FOSS communities should promote this practice amongst their contributors. This can be done via documentation, tutorials and mentoring.*

Appropriateness. The relevance of the PR is as important as the other attributes (e.g. quality, compliance, etc.). According to Tao, et al., deviation in focus or scope can lead to rejection in the FLAC and OpenAFS projects [33]. In the Linux community, patches first need to pass a gatekeeper who performs a review of the code before the code is merged. The review will fail a patch when it does not implement a relevant, working feature or bug fix, or when it does not follow guidelines of the community [16].

Although, some communities have put instruments in place to enhance the transparency of the community vision for its products, other communities still struggle to define and communicate their products' roadmaps. Some of the communities we studied maintain a vision document (VISION.md) that describes the bigger-picture goal of a project and the overall philosophy. The advantage of this approach is that community members have the opportunity to submit PRs for the vision document to propose changes to the global roadmap. This transparency allows contributors not only to understand the global vision but also to propose features that contribute to its breadth.

> ***Recommendation 3.*** *Communities should document and promote a vision document amongst their contributors. Contributors should consult available documentation and list of issues prior to submitting a PR.*

Compliance. Adherence to a computing language coding guidelines and conventions is an important element in reviewing code. Integrators consider style conformance the top factor, along with code quality, as key to acceptance. Newcomers are requested to better conform to coding standards than experienced developers, and experience in writing patches for a project leads the contributor to adhere more to the project's style [13].

Most communities we studied have guidelines for the programming languages, artifacts and processes they use. However, not all contributors follow the guidelines literally. This impacts the course of PRs evaluations. It delays the review and create tension between the author and the reviewers.

> ***Recommendation 4.*** *To avoid this type of hiccup, FOSS communities should include educational materials in their onboarding activities to create awareness of the compliance requirements. The PR template is also a good opportunity to remind contributors of their compliance obligations.*

4.3 Community

Engagement. Engagement is critical to the PR to continue to exist from inception to a decision. Lee et al. studied why one time contributors do not continue participating in projects. They cited unresponsiveness as the major negative trait [17]. Some patches are ignored because of the limited time that developers have to spend on a specific project and developers wish to maintain quality. Proper reviewing is seen as more important than addition of a feature. The contributor is expected to ping the community if reviews are delayed. Some authors become frustrated with the wait and leave the community. Others may become rude or appear overconfident or angry, and these are hallmarks for rejection [24].

Steinmacher and coauthors identified four barriers related to reception issues. The first barrier is the individual not receiving a reply to their posts. Newcomers who got a reply to their first posts were 12% more likely to post again. Other barriers are delayed answers to posts, impolite responses, and receiving too advanced or complex responses to posts. To offset these barriers, community members ought to be attentive to newcomers' reception period, sending welcoming messages, assistance, and constructive criticism, as these increase the retention of newcomers [30].

Jensen and King studied newbies first interactions with a community through a mailing list. They suggest that the feedback given to those who comment on a mailing list has a strong effect on future participation. Slightly more than 55% of new members only made one comment before leaving the community. If they did not receive a response, which occurs around 10% of the time, they were likely to leave the community. The responses to newbies were helpful nearly 70% of the time. Only 7% of the replies were polite, and the rest were rated as neutral [15].

However, social factors influence this practice [1, 13]. For example, developer's reputation affects the outcome of code review requests [4]. That is, developers in the core have a better reputation than those in the periphery. The first feedback interval is the time from the submission of a code review request until the first real review comment. Core developers have a shorter first feedback interval than peripheral developers. Reasons given may be that the developers know which teammates could best review a submission,

that their position in the core group lead reviewers to provide a less thorough review, and their prior interactions and good relationships encourage other members to prioritize their code [4].

Recommendation 5. *To ensure that every PR is equally treated at the inception and creating an engagement from the start, we recommend the implementation of the triage technique and service level agreement (SLA). Both practices have been proven to be efficient in other industries, i.e. health and incidents management for large information technology systems.*

In the health industry, triage is deciding the order of treatment for patients and casualties. In the context of FOSS PR evaluation, triage is deciding when (i.e. when the PR will be reviewed) and who (who are the reviewers of the PR). At the time of the submission (or shortly after), a community member (e.g. maintainer) should attend the PR, assign reviewers (e.g. pin reviewers with expertise in the PR topic) and set a timeframe for the evaluation process. This task can be also done by a bot.

Support. Not all contributors have the level of expertise required to present a PR of mergeable quality. Inexperienced contributors sometimes need assistance to overcome technical hurdles or to elevate their contribution to meet community requirements.

Steinmacher, et al. suggest that the reasons for nonacceptance varied, they identified 19 reasons for rejection of PRs named by quasi-contributors. The most common reason was that someone else made a similar PR that was accepted. Next frequent reason was a mismatch between developer and team's vision and contributor's vision. Another reason was lack of interest from integrators, and another was that the community was not perceived as being supportive enough [31].

Support is the essence of communitarian philosophy [22]. It is surprising that contributors do not always receive the support they need. This could be enhanced by encouraging contributors to be more assertive about it. This could be as simple as an emoji that signals to the community the need for assistance. In the human aspect of the process, communities should encourage reviewers to be warm, welcoming and supportive. The tool supporting the process (e.g. GitHub) could be also enhanced to encourage support. This can be, for instance, by implementing a "Support" feature in the contributor profile similar to the "Stars" and "Followers" features. Contributors can vote up or down reviewers when they are being supportive or unsupportive. This shall showcase reviewers' supportive attitude in their GitHub profile. This may encourage reviewers to be more supportive during the review process.

Recommendation 6. *Communities should advise their reviewers to be supportive and contributors to be assertive and ask for help when needed. The supporting platform (e.g. GitHub) can introduce profile attributes to encourage reviewers to be supportive.*

4.4 Maintainer

Decision. In most communities we studied, maintainers have the commit privileges and the final decision on the destiny of the PR. Ultimately, once the reviewers' feedback is addressed and the reviewers approve the PR, then the maintainers determines its fate. It is expected from the maintainer (or committer) to follow the recommendations of the reviewers. However, it is not always the case. Maintainers override the reviewers decisions occasionally.

The decision is the conclusion (i.e. close or merge) reached after consideration. This event is critical to the contributor' experience. She or he may experience disappointment (e.g. if the PR is closed) or a fulfilment and achievement (e.g. if the PR is merged). Our conclusion is decision should be based on technical grounds, irrespective of what the decision is. Technical grounds implies that the PR meets the community requirements on quality, appropriateness, compliance with community standards (e.g. programming languages guidelines) and other software engineering principles (e.g. passing test, maintainability, etc.).

Every time a person evaluates another's work, the social tensions that arise can produce negative emotions. In OpenStack, participants were asked about fairness. Twenty-four percent of participants subject to reviews stated that they are treated unfairly occasionally and fifteen percent feel they are treated unfairly often. Reviewers who were questioned stated that they conduct reviews fairly (60%), but some stated that they conduct reviews unfairly occasionally (40%) [8].

It would be naive to assume that a set of good practices would eliminate conflicts, rudeness and clashes of values and norms. However, implementing these practices will make a community stand out from other communities and consequently attracting more contributors. Implementing and sharing good practices nurtures a culture of quality and values.

Recommendation 7. *Decisions should be on technical grounds. The rational behind the decision must be communicated clearly and professionally. Contributors should be given the opportunity to appeal decisions deemed unfair.*

5 RELATED WORK

Several existing works have identified factors affecting acceptance of pull requests in addition to findings discussed in this study. Soares, et al. found that the most relevant factors were quality of source code, including code style. If the code is in a mergeable state, it is more likely to be accepted. In addition, project fit, technical fit and inclusion of tests were important positive factors [27]. This concurs with our findings. We identified that keeping the discussion on the technical grounds of the PR and obeying by the community principles are good PRs evaluation practices.

Soares and coauthors identified the level of the experience of the contributor as relevant to the rejection of the contribution. A first PR by a junior contributor is more likely to be rejected. The size of the pull request affects rejection too. Longer pull requests with several commits are more likely to be rejected as well. Pull requests that modify several files are most likely to be rejected, and pull requests without documentation have a smaller change of acceptance [28]. This is inline with our findings. We suggested that PR simplicity is a quality that enhances the chance of a PR being accepted.

While some believe that technical merit, which includes correctness, scope, style, design choices, and priorities is the only factor that influences contributions' acceptance in open source projects, it is known that social factors affect acceptance. Tsay, et al. report that contribution acceptance is higher for submitters with existing

relationships with core members of a project [35]. Our subjects suggest that decision to merge based on social connections is not a good practice: fair evaluations are based on technical merits.

Other social factors that affect acceptance of a pull request are personality and interaction style. Marlow and coauthors observed that developers often make stereotypical judgments about a contributor. These judgements include a belief about a user's coding ability and his or her personality. They found that developers view contributors as newcomers or competent peers. Newcomers are distinguished with lack of activities. Competent peers have a breadth and depth of the project and languages that they use. Personality assessments are made based on arguments or rudeness in posting, which often reveals an uncooperative attitude or arrogance [18]. We also find some good practices of social nature. For example, support for the contributor and a good atmosphere during the course of the evaluation are signs of a healthy evaluation process.

6 CONCLUSION

Social coding environments are changing the way software is created in FOSS communities. Incoming PRs can be rejected or asked for resubmission after improvement. Both scenarios interrupt the workflow of patch writers and reviewers, increase their workload, and potentially delay the general development process. We identified a set of good practices for FOSS communities to document, and implement to enhance the contributor's experience. We believe the implementation of these practices will distinguish communities for being a welcoming and friendly environment. This may attract and retain more contributors.

Good experience is definitely not about eradicating rejections. Rejections will remain an inherent part of the process. But a rejection after a fair, engaging and positive experience is not as demotivating as a rejection after an experience filled with setbacks. Contributions carry an enthusiasm and a motivation with it. The community has an obligation to reward this enthusiasm. This exchange does not require quantification and measurement. In contrast with our present economy system, which is based on quantifiable and measurable exchange transactions using money as unit for measurement. Even though, there is an inherent "moral" obligation to reciprocate the gesture of giving, communities should nurture this unselfish behavior, especially when the participation is based on internal satisfaction and self-enjoyment, by ensuring the contributor experience is positive. This is key for their sustainability and growth.

REFERENCES

[1] M. Y Allaho and W. Lee. 2013. Analyzing the social ties and structure of contributors in open source software community. In *ASONAM*.
[2] O. Baysal, R. Holmes, and M. Godfrey. 2012. Mining usage data and development artifacts. In *MSR*.
[3] O. Baysal, O. Kononenko, R. Holmes, and M. W Godfrey. 2013. The influence of non-technical factors on code review. In *WCRE*.
[4] A. Bosu and J. C Carver. 2014. Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In *ESEM*.
[5] L. Carroll. 2011. *Alice's adventures in wonderland*. Broadview Press.
[6] J. Coelho and M. Valente. 2017. Why modern open source projects fail. In *FSE*.
[7] N. K Denzin. 2017. *Sociological methods: A sourcebook*. Routledge.
[8] D. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue. 2018. "Was My Contribution Fairly Reviewed?" A Framework to Study the Perception of Fairness in Modern Code Reviews. In *ICSE*.
[9] M. Gharehyazie, D. Posnett, B. Vasilescu, and V. Filkov. 2015. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empirical Software Engineering* 20, 5 (2015).
[10] G. Gousios, M. Storey, and A. Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *ICSE*.
[11] G. Gousios, A. Zaidman, M. Storey, and A. Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *ICSE*.
[12] E. G. Guba and Y. S. Lincoln. 1985. Naturalistic inquiry (Vol. 75). *Beverly Hills, CA: Sage* (1985).
[13] V. Hellendoorn, P. T Devanbu, and A. Bacchelli. 2015. Will they like this?: Evaluating code contributions with language models. In *MSR*.
[14] P. Hinds and C. McGrath. 2006. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *CSCW*.
[15] C. Jensen, S. King, and V. Kuechler. 2011. Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists. In *HICSS*.
[16] Y. Jiang, B. Adams, and D. M German. 2013. Will my patch make it? and how fast?: Case study on the Linux kernel. In *MSR*.
[17] A. Lee, J. C Carver, and A. Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *ICSE*.
[18] J. Marlow, L. Dabbish, and J. Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. In *CSCW*.
[19] M. B Miles, A. M Huberman, and J. Saldana. 2014. *Qualitative data analysis: A methods sourcebook. 3rd.* Thousand Oaks, CA: Sage.
[20] R. Padhye, S. Mani, and V. Sinha. 2014. A study of external community contribution to open-source projects on GitHub. In *MSR*.
[21] R. E Potter and P. A Balthazard. 2002. Virtual team interaction styles: Assessment and effects. *International Journal of Human-Computer Studies* 56, 4 (2002).
[22] Rory Ridley-Duff. 2007. Communitarian perspectives on social enterprise. *Corporate governance: an international review* 15, 2 (2007).
[23] P. C Rigby, D. M German, and M. Storey. 2008. Open source software peer review practices: a case study of the apache server. In *ICSE*.
[24] P. C Rigby and M. Storey. 2011. Understanding broadcast based peer review on open source software projects. In *ICSE*.
[25] C. Robson and K. McCartan. 2016. *Real world research*. John Wiley & Sons.
[26] A. Schröter, J. Aranda, D. Damian, and I. Kwan. 2012. To talk or not to talk: factors that influence communication around changesets. In *CSCW*.
[27] D. Soares, M. de Lima Júnior, L. Murta, and A. Plastino. 2015. Acceptance factors of pull requests in open-source projects. In *SAC*.
[28] D. Soares, M. L de Lima Júnior, L. Murta, and A. Plastino. 2015. Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates. In *ICMLA*.
[29] I. Steinmacher, T. Conte, and M. Gerosa. 2015. Understanding and supporting the choice of an appropriate task to start with in open source software communities. In *HICSS*.
[30] I. Steinmacher, T. Conte, M. Gerosa, and D. Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *CSCW*.
[31] I. Steinmacher, G. Pinto, I. Wiese, and M. Gerosa. 2018. Almost there: A study on quasi-contributors in open-source software projects. In *ICSE*.
[32] I. Steinmacher, I. Wiese, A. Chaves, and M. Gerosa. 2013. Why do newcomers abandon open source software projects?. In *CHASE*.
[33] Y. Tao, D. Han, and S. Kim. 2014. Writing acceptable patches: An empirical study of open source project patches. In *ICSME*.
[34] S. J Tracy. 2010. Qualitative quality: Eight "big-tent" criteria for excellent qualitative research. *Qualitative inquiry* 16, 10 (2010).
[35] J. Tsay, L. Dabbish, and J. Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *FSE*.
[36] P. Weißgerber, D. Neu, and S. Diehl. 2008. Small patches get in!. In *MSR*.

G

Appendix G: Pull Request Survey

IT University of Copenhagen

This is a survey set by the IT University of Copenhagen. The aim of the survey is to understand the pull requests assessment process in open source software communities. We are a group of researchers from the IT University of Copenhagen. We part of a quality research group called SQUARE (https://square.itu.dk). The group aims at enhancing our understanding of building quality software. This research is led by Professor Andrzej Wąsowski (http://www.itu.dk/~wasowski), Associate Professor Marisa Leavitt Cohn, and PhD Fellow Adam Alami. This project is supported by the Horizon 2020 initiative. If you interrupted during the survey, you can resume later. If you have any question, please contact Adam Alami at: {ADMINEMAIL}

Thanks for participating in this survey. The results will be shared with your community by the end of Sept. 2019. We will post the result in your community forum or mailing list.

There are 15 questions in this survey.

Your role in your community

Please, tell us what is your role in your community?

Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both. *

Please select from 1 to 2 answers.
Please choose **all** that apply:

- [ ] Contributor (Authored and submitted pull requests)
- [ ] Maintainer (I have the responsibility of merging PRs in my community)
- [ ] Not an open source contributor (neither a contributor nor a maintainer)

Your community

Your current community.

Which community are you currently contributing to? Please, select a community you predominitaly contribute to. This community will be referred to as "My community", in the rest of the survey. *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please select one answer
Please choose **all** that apply:

- [ ] FOSSASIA
- [ ] Odoo
- [ ] DuckDuckGo
- [ ] Linux Kernel

- [ ] Coala
- [ ] ROS
- [ ] Plone
- [ ] ReactJS

- [ ] AngularJS
- [ ] NodeJS
- [ ] OpenGenus
- [ ] Mozilla

- [ ] OpenSUSE
- [ ] jQuery
- [ ] Apache

- [ ] Other:

Main question for maintainer

Main question for maintainer

As a maintainer, in your community, how strongly do you agree with the statements listed below? *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please select 11 answers
Please choose the appropriate response for each item:

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
In general I say no to most pull requests (PR)/patches. The contributor has to be persistent and prove that the PR/patch worth evaluating.	◯	◯	◯	◯	◯
I don't consider a pull request/patch, unless I trust the contributor.	◯	◯	◯	◯	◯
I don't consider a pull request/patch, unless the contributor is reliable.	◯	◯	◯	◯	◯
I don't consider a pull request/patch, unless I have a strong relationship with the contributor.	◯	◯	◯	◯	◯

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
I assess every pull request/patch in the same manner irrespective of the contributor.	◯	◯	◯	◯	◯
I assess pull requests/patches purely on technical grounds.	◯	◯	◯	◯	◯
I never say no to a pull request/patch. If the quality of the PR/patch is not mergeable, then I mentor the contributor to elevate his/her PR/patch to a mergeable state.	◯	◯	◯	◯	◯
My subjective assessment of quality is more important than the consensus of the reviewers.	◯	◯	◯	◯	◯
I always follow the reviewers opinion.	◯	◯	◯	◯	◯
I do not hesitate to confront well established community members during the review, if it is based on technical grounds.	◯	◯	◯	◯	◯
I'm careful during the review process. I don't like to alienate contributors who don't handle rejections.	◯	◯	◯	◯	◯

Main question for contributors

Main question for contributors

As a contributor, in your community, how strongly do you agree with the statements listed below? *

Only answer this question if the following conditions are met:
Answer was 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please select 5 answers
Please choose the appropriate response for each item:

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
I prefer to be declined by default, when I submit my pull request/my patch.	◯	◯	◯	◯	◯
My pull request/My patch shouldn't be assessed until I prove my commitment to the community.	◯	◯	◯	◯	◯
My pull request/My patch shouldn't be assessed until I prove my trustworthiness to the community.	◯	◯	◯	◯	◯
I prefer an assessment of my pull request/my patch based on technical ground, irrespective of my status in the community.	◯	◯	◯	◯	◯

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
I prefer not to be rejected at all times; instead I should be mentored on how to make my contribution mergeable.	◯	◯	◯	◯	◯

How strongly do you agree with the statements listed below: *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please select 2 answers
Please choose the appropriate response for each item:

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
I would consider changing my community, if I strongly disagree with how PRs/patches are evaluated.	◯	◯	◯	◯	◯
I'm loyal to my community irrespective of the pull requests/patches evaluation practice in place.	◯	◯	◯	◯	◯

I have you, in the past, left or changed communities because of the pull requests/patches evaluation practice? *

Only answer this question if the following conditions are met:
Answer was 'Contributor (Authored and submitted pull requests) ' *or* 'Maintainer (I have the responsibility of merging PRs in my community)' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please choose **only one** of the following:

◯ Yes

◯ No

Please, explain the last question answer: *

Only answer this question if the following conditions are met:
Answer was 'Contributor (Authored and submitted pull requests) ' *or* 'Maintainer (I have the responsibility of merging PRs in my community)' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Yes' at question '6 [Q13]' (I have you, in the past, left or changed communities because of the pull requests/patches evaluation practice?)

Please write your answer here:

Software Engineering Principles

Software Engineering Principles

How important are the criteria listed below in reviewing pull requests/patches in your community? *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please select 7 answers
Please choose the appropriate response for each item:

	Very important	Important	Neutral	Not important	Not important at all
Code Quality (my subjective assessment of quality as a reviewer)	◯	◯	◯	◯	◯
Code modularity	◯	◯	◯	◯	◯
Maintainability (the ease with which a program code can be changed in order to: correct defects or enhance the code)	◯	◯	◯	◯	◯
Comply with the community best practices of programming languages	◯	◯	◯	◯	◯
Contain documentation	◯	◯	◯	◯	◯
Avoid technical debt	◯	◯	◯	◯	◯

	Very important	Important	Neutral	Not important	Not important at all
Pass tests	◯	◯	◯	◯	◯

Social Principles

Social Principles

How strongly do you agree with these principles in the assessment of pull requests/patches in your community? *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please select 4 answers
Please choose the appropriate response for each item:

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
Trust is more important than the contribution of the PR/patch during the assessment of PRs/patches.	◯	◯	◯	◯	◯
Past reliability of the contributor is more important than the contribution of the PR/patch during the assessment of PRs/patches.	◯	◯	◯	◯	◯
A good contributor/maintainer relationship is more important than the contribution of the PR/patch during the assessment of PRs/patches.	◯	◯	◯	◯	◯

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
I believe that contributors must be able to decouple their ego from the technical assessment of the PR/patch.	◯	◯	◯	◯	◯

Product Vision

Product Vision

How strongly do you agree with the statements below? *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

❗ Please select 2 answers
Please choose the appropriate response for each item:

	I strongly agree	I agree	Neutral	I disagree	I strongly disagree
I have rejected pull requests/patches because they do not fit within the roadmap set by the community.	◯	◯	◯	◯	◯
I have accepted pull requests/patches even though they do not fit within the roadmap set by the community.	◯	◯	◯	◯	◯

Impact

Impact

What do you think is the impact of the current pull requests/patches assessment strategy put in place in your community? *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please write your answer here:

PR example

PR example

Please, share with us the link to a pull request/patch that you think was assessed fairly? (put na if you don't have time) *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please write your answer here:

Why, in your opinion, the PR/patch (you shared with us in the last question) was assessed fairly? *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please write your answer here:

Please, share with us a link to a pull request/patch that you think was assessed unfairly? (Put NA if you don't have time) *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.) *and* Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please write your answer here:

Why, in your opinion, the PR/patch (you shared with us in the last question) was assessed unfairly? (Put NA if you don't have time) *

Only answer this question if the following conditions are met:
Answer was 'Maintainer (I have the responsibility of merging PRs in my community)' *or* 'Contributor (Authored and submitted pull requests) ' at question '1 [Q1]' (Are you an open source contributor or maintainer? Please, select the options that apply to you. E.g. if you are a contributor and a maintainer, then tick both.)

Please write your answer here:

Thanks for completing the survey. If you have any questions regarding the survey please contact Adam Alami at adaa@itu.dk
17/06/2019 – 08:32

Submit your survey.
Thank you for completing this survey.