Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Johannes Tomren Røsvik

The Motivational Differences in Commercial Open Source

A Case Study on Collaboration Between Big Tech and Volunteers

Master's thesis in Computer Science
Supervisor: Eric Monteiro
June 2021

NTNU
Norwegian University of
Science and Technology

Johannes Tomren Røsvik

The Motivational Differences in Commercial Open Source

A Case Study on Collaboration Between Big Tech and Volunteers

Master's thesis in Computer Science
Supervisor: Eric Monteiro
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Open source software (OSS) has been an increasingly important part of innovation and development in the computer science field the last few decades. The practice has historically been of mainly altruistically motivated individuals, but recently commercial companies have invested heavily in their own open source projects, becoming part of a group that traditionally is critical to commercial software development. This empirical study examines how such a diverse set of developers are able to cooperate despite differing motivations and incentives by doing a case study on Google's machine learning framework, TensorFlow.

A qualitative look at TensorFlow's developer forums show that motivational differences between companies and volunteers can have concrete implications on how software development is carried out in practice. While the open source method has enabled new forms of cooperation across otherwise distinct communities, the differences do in some cases put limitations on software features and cooperation. The results suggest that the commercial objectives of companies lead to different priorities and feature sets as noncommercial open source projects. These differences could be important to understand for participants and users of OSS, as it continues to affect an increasing share of the software developers and end users rely on.

Sammendrag

Open source programvare (OSS) har vært en viktig og økende del av innovasjon, utvikling og forskning innenfor informasjonsteknologi de siste tiårene. Metodologien har historisk bestått av hovedsakelig altruistisk motiverte deltakere, men i nyere tid har kommersielle selskaper gjort store investeringer i egne open source prosjekter, og som følge blitt del av en gruppe som tradisjonelt er kritiske til kommersiell programvareutvikling. Denne empiriske studien undersøker hvordan en så mangfoldig sammensetning av utviklere klarer å samarbeide på tross av forskjellige motivasjoner og insentiver ved å gjøre en case-studie på Googles rammeverk for maskinlæring, TensorFlow.

En kvalitativ undersøkelse av TensorFlows utviklerforum viser at motivasjonelle forskjeller mellom bedrifter og frivillige kan ha konkrete implikasjoner på hvordan programvareutvikling gjøres i praksis. Samtidig som open source metoden har gjort nye typer samarbeid mulig på tvers av ellers separate fellesskap, setter forskjellene i noen tilfeller begrensninger på funksjonalitet og samarbeid. Resultatene foreslår at de kommersielle målene til bedrifter fører til forskjellige prioriteringer og funksjonssett sammenlignet med ikke-kommersielle open source prosjekter. Disse forskjellene kan være viktig å tenke over for deltakere og brukere av open source programvare, ettersom det fortsetter å påvirke en økende andel av programvaren som utviklere og sluttbrukere er avhengige av.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The idea of sharing software freely was once a novel idea among a few idealists and academics. Today, nearly every computer, phone and website rely on open source software,[14] and it is part of "approximately 100%" of Fortune 500 companies.[7] As open source has become increasingly popular, the concept has started a paradigm shift in both commercial and noncommercial software development. It has become common for both individuals and organizations to share valued software freely online, without seeking direct profit from their work. Open source software has been competitive and innovative without relying on heavy investments from companies, which may be part of the reason why companies have started to publish their own software as open source projects.

The idea that profit driven companies publish some of their most useful software under open source licenses has become commonplace, and is increasingly the norm rather than the exception. It has become a well known way to attract users, feedback and contributions, but at the expense of giving the software away for free.

These projects function in many ways the same as the ideologically driven projects under the GNU project or the hacker culture of the 1960s, but the ethical motivations that pushed development for the activists are absent for corporations. This creates an interesting space where very differently motivated participants collaborate on the

same software, for fundamentally different reasons.

Despite these differences, open source software development continues to be one of the most important ecosystems for software innovation today, and many of the most popular open source projects are built in collaboration between volunteers and corporations. Though different motivations between different parties can lead to conflicts, motivation doesn't have to be equal in order for a project to succeed.

Little empirical research has been done to examine how motivational differences appear in developer discussions, what kinds of effects they have on commercial open source projects, and how software is able to thrive in the space despite the different motivations of participants.

Understanding "why we do what we do" is important especially when the work has an effect on almost every piece of technology we encounter. In the case of open source software, the answer to this question can be very different depending on what kind of participant is asked. With the interest of learning more about these differences, and how they have an affect on commercial open source projects, two research questions were formulated:

1. What kinds of motivational differences appear in commercial open source projects?

2. How do diverse motivations have an effect on cooperation?

This thesis will perform a qualitative analysis on TensorFlow, an open source machine learning framework developed by Google. Based on forum threads containing discussions between employees and volunteers, this study will shed light on the effect of motivational differences in commercial open source software development.

1.1 Thesis Structure

The thesis will be divided into four main parts in addition to the introduction and conclusion: Literature study, method, case study, and discussion.

- **Chapter 2, Literature Study:** Firstly, a literature study will be presented in order to provide sufficient context and background information necessary to understand state-of-the-art research in the field. The topics of history, community, and motivations in open source development will be the main focus. Since this study is an extension on a previous specialization project, Section 2.1 will be mostly reused from the previous report.[52]
- **Chapter 3, Method:** As a basis for the case study, this chapter will discuss the approach, criteria of choosing the case and the process of data collection, and analysis.
- **Chapter 4, Case Study:** Chapter 4 is a case study on Google's TensorFlow framework. The case study itself will be introduced by Section 4.1, case context, which is also mostly reused from the specialization project.[52] Section 4.2 will present the concrete findings of the case study as episodes based on TensorFlow forum threads.
- **Chapter 5, Discussion:** The knowledge from the literature study and data from the case study will be put together and presented as answers to the research questions.

Chapter 2

Literature Study

2.1 The Origins of Open Source

2.1.1 Hacker Culture

While it is hard to draw a line on where to start to tell the story of the emergence of open source software, many seem to begin in the 1960s among academic communities in the US, and the emergence of "hacker culture".

Eric S. Raymond's essay "The Cathedral and the Bazaar"[46] tells the story of how a community of computer engineers prospered at MIT in the 1960s. This is where the first occurence of the term "hacker" appears, which became a popular term to describe the community members.

The field of computer science was still in its early days, so a lot of ideas were open to explore, and new computers at MIT made it possible for more people to spend time using them. The new paradigm of computing was for the first time made accessible to researchers, waiting to be uncovered. At MIT, an emerging community of engaged academics, labeled "hackers", were dedicated to exploring the possibilities of computing. Because the nature of digital programming made the distribution of ideas and solutions easy, hackers worked as a community sharing their work freely,

becoming a hub for innovation in the field.

In this sense, the ideas of sharing and non-commercialism commonly found in academia translated well into this community where commercialization of software was not yet a topic. When the purpose of their efforts were that of innovation and research, keeping things exclusive was not of benefit to hackers.

These early days of computing and digital information sharing brewed the start of a philosophy around "hacker ethics" that was documented by Steven Levy in his book *Hackers: Heroes of the Computer Revolution.*[28] The values are centered around free access, freedom of information, decentralization, non-exclusivity and having a good impact on the world. Levy summarized these values into six beliefs, a summary that has commonly been referred to as de facto hacker ethics:

- *Access to computers and anything which might teach you something about the way the world works should be unlimited and total. Always yield to the Hands-On Imperative!*
- *All information should be free.*
- *Mistrust Authority Promote Decentralization.*
- *Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race, or position.*
- *You can create art and beauty on a computer.*
- *Computers can change your life for the better.*

Although hacker culture and ethics is a description of a past culture, its values have impacted the evolution of open software development and may today be more relevant than ever before.

2.1.2 The Story of Unix

One of the earliest projects that was developed in an open manner similar to today's open source, was the operating system Unix. The system was created at Bell Labs in the 1960s, initially as a research project. Since Bell Labs and Unix was owned by AT&T and didn't have an open source license (as such a license did not exist

yet), it was far from what is considered open source today. However, since it was considered a research project to AT&T, not commercial, the source code was shared widely among developers inside and outside of Bell Labs. This sort of transparent and open environment encouraged exploration, innovation and collaboration similarly to the one seen at MIT. And in the same way, a community of hackers started to form around Unix development independently of Bell Labs.[73]

Unix was expanded with new features and ported to new systems throughout the 1970s, at a pace and schedule that resembles typical modern open source projects. This is in contrast to most other software at the time, that had larger infrequent releases in a resemblance of the waterfall model.[50] Much of this work can be attributed to developers outside of Bell Labs, especially as universities all over the world started to use Unix, and developed improvements that were adopted back into Unix.

The free nature of the open Unix codebase also led to a number of different operating systems being developed that were based on Unix code. Most notable is the BSD (Berkeley Software Distribution), which has been the basis of several modern Unix-like operating systems today including FreeBSD, OpenBSD and Darwin.[27] Open software enabled a wide set of applications, making Unix a very influential and important software project.

The main reason for the permissive distribution of Unix code was that AT&T and Bell Labs were bound to not profit off non-telecommunication projects, so Unix was considered solely to be a research project. This all changed in 1984, when the parent organization of Bell Labs, Bell Systems, was broken up into a set of smaller companies. AT&T kept control of Bell Labs and Unix, but was after the separation allowed to enter other markets, enabling them to profit off Unix.[73] In the years after 1984, with the release of Unix System V, AT&T drastically limited the distribution of Unix code by charging more than $100,000 for a source code license.[73] This had drastic effects on the Unix community and its development. In the words of Eric S. Raymond:

> *AT&T promptly rushed to commercialize Unix System V — a move that nearly killed Unix. [...] What none of us realized at the time was that the productization of Unix would destroy the free exchanges of source code that had nurtured so much of the system's early vitality. Knowing no other model*

> *than secrecy for collecting profits from software and no other model than centralized control for developing a commercial product, AT&T clamped down hard on source-code distribution.*[47]

The act of limiting the distribution of Unix code in turn removed what was one of the major factors behind Unix's success, engaged contributors from universities and the hacker community.

This form of monetization of Unix demonstrates the most obvious way that companies can profit off software, but also how it might not always be beneficial. While Unix's design patterns can still be seen in modern operating systems today, the project's popularity is dwindling while it is being replaced by free and open source alternatives.[70] [21]

2.1.3 The Free Software Movement

Around the same time as Unix was commercialized, Richard Stallman set out to form a movement that would set the stage for what the open source community is today. In 1983 he decided that there should be a noncommercial alternative to Unix. He thought that developers should be able to access the source code of the software on their computers, and should be able to share this software as they see fit. In his initial announcement of the project, he writes:

> *I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free.*[56]

This type of software was labeled "free software", where "free" is with the meaning of freedom, not price. This ambiguity is commonly addressed by using the french term "libre" for "free as in freedom" and germanic "gratis" for "free as in free beer".[68]

Stallman founded the Free Software Foundation (FSF) as a non-profit organization backing the effort, and left his job as a software engineer at MIT to work on it full time. He started seeking funding for the development of a complete Unix-like clone named "The GNU Project", "GNU" being an recursive acronym for "GNU's not Unix!". He would publish this software under a new license, the GNU General Public License (GPL), that permitted anyone to use, modify and distribute the software as long as the

license was unchanged.[57] The central value behind the FSF and the GNU Project is freedom of software. It is what motivated Stallman's decision to start them in 1985, and still remains.[56][67]

The GNU Project grew throughout the 1980s, and became a major focus for the hacker community. By 1987 several major pieces of the GNU project were released, including an editor, compiler and debugger. Their motivation was to preserve software freedom in order to give hackers an environment free from proprietary software and commercial interests.

One important part of the GNU Project that did not get completed, was a kernel. All of the completed software still relied on a foundation of proprietary software at the kernel level. Although efforts were made to make a libre kernel, this part still remained unsolved in 1991, when Linus Torvalds started work on Linux.

Torvalds, a self taught hacker from Finland, started work on his own kernel in the early 1990s. He wanted a basic kernel that was free in terms of cost, gratis, not necessarily libre. He said that linux was a hobby project,[71] "for fun" and as an exploratory attempt to see if he could improve on pre-existing operating systems.[72]

Torvalds didn't give much thought to the philosophy of the FSF around free software, but was opposed to the idea that software should be sold for money. The earliest releases of Linux were under a short informal licence Torvalds wrote himself with the condition that Linux should be distributed for free (in terms of cost). However, the GPL was later adopted instead, which in turn provided the GNU Project its last missing piece. Several distributions of Linux and GNU software were put together in the 1990s labeling themselves as GNU/Linux. A fully free alternative to proprietary software had been created.

One of the ways in which Linux differed from the GNU project was in its collaborative nature. While Torvalds himself did most of the work in the beginning, it eventually evolved into a collaboration between thousands of developers. The model was explained in Eric S. Raymond's book "The Cathedral and the Bazaar".[46] Raymond calls it *The Bazaar model*, a methodology about keeping the entire development process public in order to give testers and developers frequent updates and insight. Raymond argues the Bazaar model decreases the amount of bugs, and helps decrease complexity,

which lead to what he calls “Linus’s Law”:

> *Given enough eyeballs, all bugs are shallow.*[46]

Linus Torvalds and the Linux community went on to make *git*, a collaboration tool to help organize the development of Linux.[10] It was released in 2005 and is today the most popular version control system for open source projects,[6] and the foundation of GitHub, GitLab and Bitbucket.

Raymonds analysis and praise for open development models had a significant impact on the industry. In 1998 the Netscape Communications Corporation announced that the code for their browser would be released to the public,[34] inspired by Raymonds work. In response to this announcement, a group of free software community members met to discuss a way to cater to the businesses that wanted to adopt open development, and the term “open source” was suggested.

> *We realized that the Netscape announcement had created a precious window of time within which we might finally be able to get the corporate world to listen to what we have to teach about the superiority of an open development process.*
>
> *We realized it was time to dump the confrontational attitude that has been associated with “free software” in the past and sell the idea strictly on the same pragmatic, business-case grounds that motivated Netscape.*[37]

The Open Source Initiative (OSI) was founded shortly after the meeting, and the term and foundation received support from large parts of the free software community. The purpose of the OSI (as of 2020) is to promote the benefits of open source and to maintain the *Open Source Definition (OSD)*.[38]

Richard Stallman was reluctant to adopt the term because of the slightly looser definition of open source compared to free software. He argues that the user’s freedom is not always preserved through open source, and that the open source philosophy could be damaging to the free software movement.[58]

Because of this disagreement, two different approaches to free / open source software were made clear. On one side, Richard Stallman and the Free Software Foundation promoted an ideological view on free software as a means to keep users in

control of their computers, and the software free from commercial intervention. On the other side, the Open Source Initiative viewed open source software as a beneficial development model, also in the context of commercial development. For the OSI, the act of sharing source code was less in terms of freedom, and more in terms of practical utility.

2.2 Community

Looking at the differences between OSS development and conventional development, the community aspects of OSS is one of the most important distinctions. Raymond [46] argues that the distributed community was the key to Linux' success, and the distributed community was an enabler for new kinds of development processes that commercial projects lacked. Thus, studies on the community aspects of OSS are important to understanding a key part of modern software development. Community activities are clearly part of a company's motivation when embracing an open source development approach, since a project without community is just code, and code that is not maintained will become less useful and depreciate in value as technologies change.

Projects like Unix and Linux demonstrate how a project with productive collaboration between its maintainers and users is able to produce innovative tools for the public. Communities that provide a place for productive innovation have been labeled *communities of practice (CoP)*, coined in 1991 by Lave [26].

> *Communities of practice are groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly.* [75]

Thus, a CoP has three main characteristics: a domain, community and practice. CoP is value creation through knowledge work within a community. According to Wenger [74], a CoP can educate, encourage and support work within a community.

Like communities of practice, open source communities are interactive and have a common field that the collective works to advance and learn about. In open source, a project cultivates such a community of developers within a field by leaving it open

to those with the interest and dedication to join. As open source projects have a notoriously open culture; the sharing of experience and domain knowledge is encouraged. The open source community enables innovation within their field, specifically by building a piece of software together. In the case of Linux, developers that have experience within operating systems can participate in the community by sharing their knowledge with the community and contributing concrete solutions to the codebase that further progresses the field.

Bechky [5] expands on the concept of communities of practice and argues that the people involved do not need to be completely uniform. A CoP can have distinct subgroups with very different characteristics, as long as they have a certain common ground that enables a shared understanding of the field. In the study, certain shared physical objects are able to satisfy the needs of distinct groups by bridging a knowledge gap. These objects are called *boundary objects*: Objects that enable distinct groups to have a shared understanding and practice, despite different backgrounds.

The concept of boundary objects was introduced by Susan Leigh Star and James R. Griesemer [59]:

> *[Boundary objects are] an analytic concept of those scientific objects which both inhabit several intersecting social worlds [...] and satisfy the informational requirements of each of them. [...] The creation and management of boundary objects is key in developing and maintaining coherence across intersecting social worlds.*

Commercial open source projects have several of the same properties, as they act as shared objects for participants with different backgrounds and motivations. It is maintained in cooperation between companies and volunteers, providing information that enables coherence where there previously was none.

2.2.1 Types of Community Members

It would here make sense to make a distinction between the different types of community members. Eghbal [15] categorizes the people involved in open source projects as *maintainers*, *contributors* and *users*. When talking about commercial motivation, it

can in addition be useful to distinguish between maintainers and owners, whereas in noncommercial projects the owners are usually also maintainers.

- **Users**: Consumers of the project that only interact with the project by downloading and using the end-product. They may have used the forums and documentation to learn about usage, but haven't made themselves known on these forums.
- **Contributiors**: Participants that have made themselves known through a post to an issue tracker, forum or mailing list, or have suggested concrete changes to the project. This can range from casual one-time contributors to those who regularly participate in discussions and provide code contributions.
- **Maintainers**: The most active contributors with the most responsibility, access and ruling power. On commercial open source projects, this is typically a paid developer team at the owner organization. The maintainers are together with the owner responsible for the long term progression of the project.
- **Owner**: The single person or organization that created the project, set the license and has final say in important decisions.

This study is mainly focused on the interaction between commercial maintainers, and volunteer contributors. Commercial maintainers are typically the employees of the owner organization that are paid to be part of the development team, and are considered representatives of their employer. Other contributors outside of the internal team participate on their own accord, presumably with different motivations and goals than employees. Examining the motivation of each group helps in identifying differences and understanding potential conflicts.

The role and importance of outside contributors vary from project to project. Some projects are bazaar-like in their methodologies, with a large set of engaged contributors all collaborating to produce software together. Examples of these include Linux,[29] Rust[51] and Node.js.[35] In other cases, there might be a single person or team that manages the vast majority of project tasks. In these cases, the contributors are the

least prominent group, and the project is a matter of maintainers publishing a product to users, with much less interaction than in a collaborator heavy project. Examples of these are Babel[3] and Clojure.[11]

A grouping of these types of projects have been proposed by Eghbal[15], where projects with high contributor and user count are labeled "federations" and low contributor count and high user count are labeled "stadiums". In addition, projects with a high contributor count and low user count are labeled "clubs" and low contributor count and low user count are labeled "toys". A full table of categories are shown in Table 2.1.

	Many users	**Few users**
Many contributors	"Federations"	"Clubs"
Few contributors	"Stadiums"	"Toys"

Table 2.1: Types of OSS projects classified by user and contributor count

Because of these differences in degrees of community involvement, a study of a federation project may not apply to a stadium. In highly distributed OSS projects where contributions are the main driver for project progression, maintaining an active and satisfied contributor community may be detrimental to the project's ability to survive, while in a stadium, maintainers need only be worried about their users.

2.2.2 Joining an Open Source Project

Although OSS projects don't have a hiring process like conventional software development, there are norms and expectations of contributors that want to take part. Krogh et al.[25] examines a case of structure in an open source community and how it functions in terms of joining, rewards and specializations.

The study shows that there are significant barriers for people to join a project, and become part of the community. A "joining script" is often followed where certain amounts of engagement and level of competence has to be shown before a new developer is allowed the same access as established contributors. Such a script creates a

certain barrier for entry that limits the number of people in the core community to a set of dedicated volunteers, and means there are substantial costs to becoming part of a core developer group.

The type of activity from a joiner is usually different from other community members. Activities like posting technical questions, reporting bugs and proposing bug fixes are more common among joiners. This reflects the lower project knowledge among joiners, limiting the type of contributions. As a joiner becomes more familiar with the project, they are likely to acquire knowledge about a specialized part of a project, and make more advanced contributions to a narrower area. In large projects, only maintainers have enough general knowledge to understand the entire project and how each module works in relation to others.

However, if a joiner has prior knowledge that can be applied in a new project, OSS projects may receive a "feature gift". A feature gift is a larger new module or feature that the contributor was able to contribute with relative ease because of previous experience. Such a contribution is also a form of specialization, where a new community member becomes a specialist within their own module. In this way, a OSS project evolves with the help of many domain specialists that are managed by a few generalist maintainers.

Practically speaking, a joiner usually doesn't have the necessary access to make changes to the project directly, which is commonly solved by the joiner making changes to a copy of the project instead. In the case of a GitHub repository, this is done by first creating a copy, "fork", of the project, then making changes to that copy. The changes are then submitted as a "pull request" to the original project, which maintainers can accept or reject to become part of the original code base. This way, bug fixes, new features and improvements can in part be outsourced to volunteer developers.

Another important form of community involvement is in the form of discussion on issue submissions, forum posts and mailing list. This kind of activity can be both a resource and resource drain for development teams. While bug reports, feedback and feature suggestions can be helpful for development, developers on popular projects are sometimes overwhelmed with posts from users that want support with usage issues.[15] Managing such requests from users can be a time sink if there are too many low quality posts. However, given the fact it has become commonplace for companies to open up

these channels, it seems that companies consider it a worthwhile effort overall.

2.3 Motivations in Open Source Projects

In a study looking at the motivations of commercial open source, it is also important to consider the motivations of its community, and how they participate in a OSS development setting. If Raymond was right in his analysis, this can be essential to the project's ability to succeed, and a valuable asset to a commercial owner. To understand how such a community can come about and function, we need to recognize why each member is participating.

While there may be drivers that have some part of all participants' reasoning for contributing, motivations for different participants are diverse and vary from person to person. As noted in Krogh et al. [25], community related benefits are an important part of individual motivations, but in commercial projects where part of the community are paid employees, these factors may be less of a priority from some.

Looking at the motivations of different kinds of participants in OSS projects will reveal that there are large differences in motivations between different types of actors. Despite these differences, there have been plenty of successful projects like this in the past, but friction between community members because of the different backgrounds is certainly a possibility. Either way, common motivations within each category of participants will be useful to understand when examining a case later on.

Both users and paid maintainers have clear monetary incentives for taking part. Users can benefit from costless software, while paid maintainers can treat their contributions like any other day job. What is less clear is why companies share their software for free, then spend resources maintaining it, and also why volunteer contributors use their spare time to help such an organization without payment.

Hippel and Krogh [23] categorizes open source investments as part of the "private-collective" innovation model, where private-collective innovation is an alternative to conventional for-profit "private investment" innovation and charitable "collective action" innovation. While private investment projects' purpose is to provide value to the private owner, and collective action is to serve the interest of the general public,

open source software's private-collective model seems to have both commercial and public advantages. Open source software has proven that open source is a viable form of innovation that is neither profit driven nor charity, rather somewhere in between.

2.3.1 Motivations of Contributors

One of the key advantages of having a project be open source, is the ability for the community to contribute concrete changes to the code. The fact that the code is available makes it possible for anyone with the necessary skills to download the project code, make changes or additions, then suggest their changes to the project's maintainers.

Broadly speaking, two categories of motivations for contributors have been used in previous studies, intrinsic and extrinsic. Intrinsic motivations are commonly described as "doing something for its own sake" and can be objectives like learning, enjoyment or morale. Extrinsic motivations are commonly described as "a means to an end" and are typically payment or concrete perks from contributing, including increased use-value.[49] Though it can be useful to categorise types of motivations, the dualistic approach of intrinsic and extrinsic motivations has been criticized for lack of nuance.[48] Reputation is one of the motivators that doesn't neatly fit as purely intrinsic or extrinsic, but is rooted in both.

Although maybe not the most important, extrinsic motivations are a significant driver for contributors. Since most contributors also are users of the software, improving the software also adds to the use-value for the contributor. When software is a non-rival commodity and the code is public, developers are able to improve the software they use themselves, and there is usually nothing for them to lose by sharing the improvements.[31]

When use-value is the main motivator for a contributor, it's common to leave once their issues are resolved, or the feature is complete. However, some choose to stay for longer and expand their contributions beyond their own needs. For some, contributing becomes a hobby. These hobbyist contributors can be critical to sustaining open source projects in the long term.[55]

While direct monetary rewards are uncommon outside of the most active maintainers, open source participation can lead to employment opportunities outside of the project itself. A prominent position in an open source project, can serve as a way to demonstrate knowledge and competence to future employers.

Raymond [45] explains how OSS communities represent a fundamentally different kind of economy to our traditional exchange based economy. This kind of culture is due to an abundance of resources in software development. Disk space and computing power is abundant, so the only cost of participating is time and effort. Usage of time and effort can be exchanged for a new kind of currency, *reputation*. Reputation gained through public mentions in changelogs or positions on contributor lists becomes the currency in an ecosystem with abundant resources. In other words, those with time and knowledge to contribute, do so in exchange for what is mainly a social benefit. Raymond explains this like so:

> *Abundance makes command relationships difficult to sustain and exchange relationships an almost pointless game. In gift cultures, social status is determined not by what you control but by what you give away.* [45]

While extrinsic motivations show some benefits to both owners and users in a private-collective model, a purely extrinsic perspective falls short in justifying why someone would undergo the effort required by steady maintenance. Use-value and reputation only goes so far to motivate contributors. So to understand why contributors stay around and contribute independently of their own benefit requires a look at the intrinsic motivations of community members.

Eghbal [15] makes comparisons between an OSS community and other online communities on social networks. For some active participants in online social contexts where there is a high sense of community, the content produced becomes less important than the community itself. In OSS projects, the product of a CoP is considered highly valuable for users, but the contributors who return again and again to forums and contribute code repeatedly, don't do so because of the code, but because of a sense of belonging. Eghbal argues that platforms for open source projects have become more like a social platform than a development tool.

While other online communities of practice provide the sense of belonging, someone to share interests with and a place to gather knowledge, the value of open source projects in modern software development adds new meaning to the community. The value that to users gives OSS communities purpose and esteem. The challenge of creating good solutions can be a fun challenge and accomplishment to participants.

2.3.2 Motivations of (Commercial) Owners

While both extrinsic and intrinsic motivations are important for a volunteer open source contributor, the same cannot be said for commercial organisations. Given that companies follow the definition of *commercial*, extrinsic motivations are the main driver.

> ***commercial**, adjective: making or intended to make a profit*[42]

If an organization is commercial as opposed to non-profit, the assumption is that every activity the company spends time and resources on, is at the core motivated by generating a profit. But though monetary output from open source may be the foundation for an organisation's activities, a more nuanced view is needed to understand the multitude of ways that it is reflected in an organisation's choices. The road to commercial success are many, and some are less direct than selling licenses or services.

As opposed to the simple profit-driven view of companies, Bonaccorsi and Rossi [8] proposes a higher-level taxonomy of commercial motivations for open source contributions with three main categories: Economic, social, and technological. Social and technological motives show that organisations can have goals that are not directly monetary, but grounded in commercialism nonetheless.

In most cases the monetary justifications for commercial open source projects are well known. There has been thorough research on different kinds of business models surrounding commercial open source projects that in one way or another makes it possible to profit off OSS in other ways than selling the software directly.

Many of the common methods are variations of the "loss leader" strategy.[4] By giving away some software for free, users are more likely to become customers of other related products or services that the owner provides. Variations of these includes

but are not limited to:

- **Support services**: By selling premium technical support and help to users that are willing to pay for it. Red Hat is one example of a business that sells "convenience" of supply and support of open source licensed software.[76]
- **Dual licensing**: By releasing software both under a copyleft license and a proprietary one, the users that want to further develop the software without the copyleft license will have to pay for the proprietary one. As an example, the MySQL database is available under two versions: "MySQL Community" under the GPL license and MySQL Enterprise with a commercial license for customers that are unable to comply with the GPL.
- **Related products**: A large user base can help to draw attention to related proprietary products, or even create demand for a service. One example of this is MonogDB which develops a database and sells hosting of the database separately.

All of these business models are at the core based on the same principle: Use an open source license to get the software in the hands of the masses, then profit off the users that encounter limits, like licensing issues or the need for support. In a sense, the open source model works as a marketing tool for companies, and a way to outcompete proprietary solutions.

When MongoDB founder Eliot Horowitz was asked why he didn't make MongoDB closed source, he said:

> *No one would have used it.*[22]

Whether or not a company "cares" about the social value and altruism of their work, the ripple effect of ignoring the norms of OSS development could affect the success of the project. If the company acts against the interest of it's community, contributors may leave and users may look for competing solutions. This way, it seems that companies are forced to adhere to the values of OSS development even though their commercial nature may conflict with it.[41] But if the company behaves well as stewards of an OSS project, the community can reward the company in return.

So when the community becomes an important asset to the company, the community gets to influence the companies to adapt their norms and values. An unwritten contract is established between owners and volunteers based on trust and the social norms of the OSS community.[8]

The technological advantages of open source software have been the topic of many previous studies. Because of the above mentioned motivations of individuals, open source attracts the attention of private contributors that bring their expertise to the project, increasing its value for every party. Hippel and Krogh[23] labelled this the "Private-Collective" innovation model, a hybrid between charitable and commercial innovation. This model contains several of the advantages of both models, while serving as an advantage to both the general public and the company. This way, feedback and contributions from the public are certainly one of the main selling points for companies entering open source development.

Chapter 3

Method

The purpose of this thesis is to provide more knowledge to the commercial open source research field. With a focus on motivation in commercial projects, a qualitative look at one individual case was chosen as the approach to gather material for analysis within the time frame of the thesis.

Specifically, this was done through a case study on TensorFlow, an open source machine learning toolkit released and maintained by Google. A public forum will be examined, and the results will be compared to previous research in order to discuss what parts of the findings are new and interesting.

This chapter will describe the methods around research, choice of case, data gathering and analysis that was used for this thesis.

3.1 Approach

In order to answer the research questions outlined in the introduction, a case study was chosen as the most appropriate method. While there are a lot of studies data already in the field of open source software development, the thesis was in need of further empirical data in order to facilitate analysis. Especially since the choice of focusing on differences in open source motivations was made partially because of the

lack of previous research and data.

A case study as a research method is well suited to provide empirical data considering the availability and abundance of information on open source projects. But for the results of the case study to have any meaningful impact, it needed to be done in the context of previous literature within the field. A literature review was therefore done as the first step in the process, to establish a foundation for any results to build upon. The case study and literature review are the ingredients of Chapter 5, Discussion, where the results of the literature and case study will be put together. The findings in the case study will be presented in the light of the literature study in the hopes of uncovering new insights.

The literature review is a large part of the study, and what forms the basis and background of any results. As the first step in the study, it gathers knowledge about the field of open source software development, in order to facilitate a state-of-the-art study. Understanding the current state of similar research was considered an important precondition for the study to contribute further field knowledge.

A fitting commercial open source project, TensorFlow, was selected as the subject of the following case study for the reasons explained in Section 3.2. In general, the case was chosen for its ability to provide insight into commercial open source and motivations.

The type of approach for the study is mostly qualitative instead of quantitative. This decision was made mainly because of the type of research questions, as qualitative methods were best suited to answer them. The answers to "how?" type research questions are not easily "measured", but rather interpreted through in-depth examination for which a qualitative study is more suitable. A nuanced understanding of developer interactions is needed to understand the issues, since the way motivations show themselves in forum posts are just implied, not explicitly written.

This way, the study is able to dig deep enough in a few places to identify these unwritten motivations, and to properly understand their effects on collaboration, even though it is limited to just a single open source project.

However, in Section 4.1, quantitative methods to provide context and background understanding of the case will be used in addition to the qualitative analysis. Quanti-

tative metrics like participation count, and activity amount will be used to describe scale and categorize participants. Thus quantitative results are not directly part of the results, but rather part of understanding their context.

3.2 Choosing a Case

Since the case study was based on the findings from a single project, selecting a suitable case was important. There were several factors that rendered a case uninteresting for this kind of study, so a set of criteria needed to be established. Thankfully, there are a lot of commercial open source projects to choose from, and more likely than not, several would fit the criteria.

Firstly, the case needed to be able to support the study practically by having enough information of the right kind to examine.

1. **Size:** The size of the project should be large enough to require collaboration between several maintainers, which will show more of the dynamics of open source software development. The scope should also be significant enough that there are issues raised about new features and development direction.

2. **Age:** The age of the project should be at least a few years, so that there is a significant amount of historical data to examine, and the effect of discussions become apparent. But in order to keep the case study relevant to modern open source development, it should not be too old either. Therefore, the project should optimally be between three to ten years old, in order to be modern but properly initiated.

3. **Well documented:** This study is dependent on access to discussions through forums or mailing lists that are active enough to gain insight. This is the data that will support the study, so there needs to be plenty.

Secondly, the case needs to fit the research topics of the study. Some thematic criteria were established.

4. **Licence:** The project should be open source software, meaning that it is published under a Open Source Initiative approved license[40] that follows the Open Source Definition.[39]

5. **Commercially owned:** The project should be owned and maintained by a commercial organization that for some reason is choosing open source development as a business strategy.

6. **Generalizable:** In order to be somewhat generalizable, the project and its owner should not have special attributes that makes the case completely distinct from other actors in the field. If the case is similar to several other projects, the findings are more likely to be general truths rather than unique situations.

These criteria did filter out a significant portion of commercial open source projects. For example, projects owned by Microsoft were excluded due to Microsoft's recent acquisition of GitHub,[33] a special situation that may make the project less generalizable. Projects like MongoDB[32] and Elasticsearch[16] were excluded due to noncompliant licenses. Some projects like FoundationDB[2] may have had too little forum activity to support the study.

3.2.1 The Selected Case

A project that did fit every criteria was TensorFlow.[64] TensorFlow is a large open source machine learning framework that has been available under the OSI compliant Apache 2.0 license since 2015. It is owned and maintained by Google, a company that has similar interests in open source software as many other large technology companies. Yet, it has an active community that has created thousands of forum posts with discussions.

TensorFlow's business model is less clear than many other commercial open source projects, since there is none of the usual business models like dual licensing or premium support. It is presented much like a "gift" to the machine learning community, rather than software owned by Google, which would be unexpected behavior from a profit

driven company. Either way, it makes TensorFlow an interesting case to examine closer.

Regarding its scale, TensorFlow is much larger than what is strictly necessary, with the number of participants in the thousands. Such a project could certainly provide an abundance of data, which increased the likelihood of finding situations concerning topics beyond what was found in the literature study and the field of open source research.

It is well known, actively worked on and relevant to a large number of people, including machine learning researchers. Though the end goal of this study is to find generalizable results, its implications could potentially give value to many of the stakeholders in TensorFlow specifically.

For these reasons, TensorFlow was determined to be a strong candidate, and selected as the case for this study.

3.3 Data Collection

The case part of this study will first be introduced with context around the case study itself. This was based on third party sources like news articles and forum discussions in addition to official resources by Google.

When using statements in articles, it was especially important to maintain a critical view on sources. While official sources may be the most reliable source for some metrics, statements that are mainly intended for marketing should be treated as such. The Internet Archive Wayback Machine[69] was used to access old versions of some of these pages, which showed that some content had been removed and changed over time.

The case context will also contain some quantitative analytics of the source code repository in order to understand the scale of the project and its community. By counting code changes to the source and who made the change, views that quantitatively show how development has evolved over time can be generated. A similar approach has been used by Krogh et al. [25], but since the goal of this study is different, the methods will differ as well.

The main part of this case study will be based on discussions, “issues”, within Ten-

sorFlow's source code repository on GitHub.[64] These discussions are around specific technical changes to the source code, which serves as the main channel for interaction between participants.

By April 2021, there have been more than 40.000 issues posted to the TensorFlow repository. Assuming that only discussions with more than 10 comments are worth looking into, reduces the number of to around 4.500 issues, which still is far more than a qualitative study like this is able to consider.

Though there are other sources that could have been used in this study, like mailing lists and support forums, the issues have more than enough material in a format that was well suited for filtering and sorting through.

3.4 Data Analysis

The process of turning raw data into meaningful knowledge had several stages. A central goal of the study was to use both previous literature and new data in order to generate new results, meaning there would be both inductive and deductive work. Because of this, the research questions were iteratively updated as the case study progressed.

The first part of the process was focused on previous research in order to gain understanding both foundational and state-of-the-art research within commercial open source development. A few topics around commercial participation in open source was chosen, with the main focus on the motivations and the inner workings of OSS projects.

Initially, these topics guided the search for further insight in the case. As new findings were made, the topics were slowly narrowed down in order to fit the parts of the findings that turned out to be interesting. Deductively, the literature review affected how the case was examined, then inductively the case study affected the research questions.

As the process progressed, interpretations of the case study narrowed down the study to focus on a few central areas. At this point, time was partially spent researching the case, and partially expanding the literature study as a means to explain the obser-

vations. If an observation seemed to contradict or expand upon what was available in previous literature, a potential theme for discussion was found. This process of gathering topics inductively does mean that the direction of the study was gradually changing over time, and the research is the result of a narrative guided by the observations.

These central themes were a combination of the most interesting findings in the case study and what was identified as a lack of previous studies in the literature review. This could then be presented as the final results.

The direction and central themes of the study is therefore guided both previous field knowledge, and the case itself. Previous knowledge from the literature study, as

Source Data	Derived Theme	Central Theme
"Does TF prefer to move ONNX to another repo in order to prevent better interoperability?"	Wanting to maintain users despite community wish to be interoperable reveals that there is a divide between the two parties.	Conflicting values and motivations creates a divide with tension and degraded trust between company and community.
"I know this issue might not get that much attention from Google since they have their own interests"		
"It's strange that Google ditched open OpenCL for proprietary CUDA."	Different prioritisation because of different value assessment. Using libre software (hacker ethics) vs. monetary return on investment (commercialism) creates tension.	
"I find it frustrating that people are willing to write kernels in CUDA but balk at doing it for CL."		
"I have working CPU/GPU kernels for complex FFTs [...]. Would you be up for a chat to iron out those bumps so we can add the CPU kernels?"	Cooperation on the right issues is mutually beneficial. Intrinsically motivated volunteers are willing contribute feature gifts that supplement employees work.	Within boundaries, diverse motivations and goals enables effective collaboration between contributors that lead to mutually beneficial contributions.
"Thanks to [Developer 11] and [Developer 13], I think we can call this done!"		
"Good suggestion. I can reproduce your performance data now. I will do some profile and let you know the result."	Open discourse and diverse motivations enables all parties (volunteers, employees and external companies) to collaborate effectively, and achieve goals that are in everyones best interest.	
"Thanks for all the information. Looking into this, I will update once I find the root cause."		

Table 3.1: Central themes derived from the case study

the lens through which the results are interpreted. The case and findings, in that what presents itself as noticable findings guides what kind of topics are mentioned in the discussion.

In order to systematize these results, data from the case was aggregated into a few key themes concerning open source motivations, that again could be linked to one of the two central themes of the study. The resulting table from this process is Table 3.1, which became the basis for discussion in Chapter 5.

The left column labeled “Source Data” are some of the quotes from the case study, quotes that were selected because of their relevance to better understanding motivations and cooperation. The two points in the right column are the central themes of the study, the final set of topics that the literature and case study points to. The middle column, “Derived Theme”, acts as a link between the specific source data to the general themes of the study.

As described above, the creation of the table was far from straightforward, but it serves as a concrete visualization of the relationship between case and results. The table was modified several times to reflect the current state of the study, which means that Table 3.1 shows only the final version of this process, and the final interpretation of the results.

Chapter 4

Case Study

4.1 Case Context

This study will look at the TensorFlow project by Google, an open source machine learning framework released in 2015. TensorFlow's main purpose is to help in the development of machine learning models like neural networks, and is used by developers to develop machine learning functionality into applications that run on devices that range from phones to large server farms. The history of TensorFlow development is summarized in Table 4.1.

Google is one of the major tech giants in the US, known for services like Google Search, Gmail, YouTube and Translate. The company is a subsidiary of Alphabet,[43] which as of 2020 was the fourth largest technology company in terms of revenue.[17] Most of Google's revenue comes from advertising, more than $37 billion in Q3 2020, around 80% of its total revenue.

Google has long been sharing parts of its in-house software as open source, including its mobile operating systems Android, the browser Chromium and the programming language Go. It has been an active and engaged contributor to open source software for over a decade, and have been attentive to their relationship to the community by actively participating for many years.[19] [18]

Jan. 2010	Google X is founded as a research group at Google.
Early 2011	The Google Brain team led by Andrew Y. Ng and Jeff Dean is established to focus on advancing machine learning research at Google.[30]
Jun. 2012	Markoff[30] reports *"The research project had now moved out of the Google X laboratory and was being pursued in the division that houses the company's search business and related services."*
Dec. 2012	A whitepaper on DistBelief is published,[12] Google uses the software to train large scale machine learning models for internal projects.
Feb. 2015	Astro Teller: *"Google Brain is producing in value for Google something that would be comparable to the total costs of Google X"*[13]
Nov. 2015	TensorFlow is released as an open source project based on experience gained from using DistBelief.[1]
Jan. 2017	PyTorch, a TensorFlow competitor developed by Facebook, is released as open source on GitHub.
Feb. 2017	TensorFlow 1.0 is released signaling that the framework is ready for production use.[53]
Sep. 2019	TensorFlow 2.0 is released with focus on features for ease of use, making it more accessible to less experienced users.[66]

Table 4.1: TensorFlow history timeline

Development of TensorFlow was started at the Google Brain team at Google X, a is a sub organization of Google with the purpose of performing research and development of emerging technologies that could have applications in the long term. Google X was established in 2010, and the Google Brain team was assembled in 2011 with the goal to build a machine learning system that could utilize the computing infrastructure at Google.

The initial machine learning system built by Google Brain was named *DistBelief*. The major feature of DistBelief was the creation of two parallelizable neural network algorithms that were able to run on several CPU cores and across computers. This enabled distributed training of neural networks, which made it possible to use thousands of CPU cores at once. This increased the speed at which training could happen, and allowed the use of larger models.[12] The framework was utilized throughout Google, helping in development of Google Search, advertising, Google Photos and more. In an interview with The New York Times, head of the Google Brain team, Astro Teller, said:

> *Google Brain is producing in value for Google something that would be comparable to the total costs of Google X — just that one thing we've spun out.*[13]

Although neither the revenue from Google Brain, DistBelief or TensorFlow are public, nor the budget of Google X, it should be clear that the technology has been very lucrative to Google. Considering that the quote is from the early days of the project, it can be assumed that it has had a significant return on investment for Google.

TensorFlow was developed as the next generation of DistBelief, where it was expanded to enable the use of GPUs, and portable devices with operating systems like Android and iOS. At this point, Google decided to open source the project, and November 9th 2015 the TensorFlow v0.5.0 was released on Github under the Apache 2.0 license.[63] Along with the library itself, a whitepaper named "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems" was published.[1]

On the official TensorFlow website,[62] Google addressed a question around their motivations behind open sourcing the framework. In a section labeled "Why Did Google Open Source This?", that was removed in late 2016, it says:

> *Research in [machine learning] is global and growing fast, but lacks standard tools. By sharing what we believe to be one of the best machine learning toolboxes in the world, we hope to create an open standard for exchanging research ideas and putting machine learning in products.* [60]

Although TensorFlow was not the first open source framework for machine learning, the release was considered a significant event by people in the field. [20] The framework was relatively easy to use through its Python API and had high performance because of the above mentioned support for parallelism, that was implemented in lower level languages.

The most notable competing framework to TensorFlow is PyTorch, released by Facebook in January of 2017. According to GitHub, PyTorch is used by about half the number of projects as TensorFlow (around 48.000 and 103.000 projects, respectively). [64] [44] Statistics from Google Trends can be used to give an indication of relative interest between TensorFlow and PyTorch. Figure 4.1 show that PyTorch is historically a less used search term, but recently they have attracted similar amounts of interest globally.

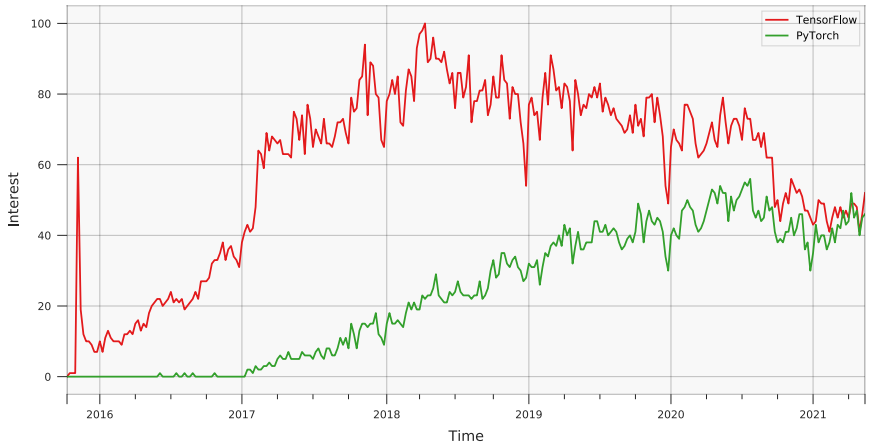

Figure 4.1: Google Trends data showing interest over time of TensorFlow and Pytorch as search terms on Google. Relative to the point with most interest at 100.

4.1.1 Communication Channels

To provide platforms for communication within the TensorFlow community, public mailing lists were created. The first, and most active among these, is "TensorFlow discuss" a list for general community discussion. As of November 2020, this list has more than 4000 threads. Specific mailing lists for development, documentation, testing and announcements have been created in order to categorize the large volume of messages.[61] These mailing lists are:

- *discuss@tensorflow.org* is the most active, and general of the lists. It is mostly used to discuss and provide support regarding usage of TensorFlow, rather than proposing changes or bugs in the framework. Common submissions to the mailing list are questions on how to apply the framework in a specific context, solving specific machine learning problems or newcomers asking for help to get started.
- *developers@tensorflow.org* are for discussion among TensorFlow contributors. Common messages are questions about current implementations and help with problem solving issues during development.
- *announce@tensorflow.org* is used as a low volume channel to announce releases and changes to TensorFlow.

There are also less active mailing lists for testing and documentation, in addition to lists for related topics like TensorFlow Hub, Keras and add-ons.

However, in terms of communication volume, the GitHub repository is the main platform for the community. Since December 2015, it has been the main channel for feature suggestions, bug reports and pull requests. GitHub is where most of the public development discussions have taken place, both for Google employees and external contributors.

4.1.2 Activity Metrics

Looking at the repository tells a lot about how active the development of tensorflow has been since 2015. As of November 2020, more than 27.000 issues and 15.000 pull

requests have been opened on GitHub. Over two thirds of the pull requests have been accepted and merged into the codebase. Out of these, there are 2700 different contributions. Judging by these numbers and TensorFlow's high popularity, the project can be categorized as a "Federation" according to Table 2.1.

Judging by the most active pull requests and issues on the GitHub project, the project has been open to feature suggestions from community members outside of the team of Google employees. Several of the largest contributions (in terms of lines of code in a single pull request), have come from contributors outside of Google.

One indicator for the size TensorFlow community, are the email addresses contributors use in their code commits. Since every commit to a Git project has to be linked to an email address, one can count every unique email address that has made a commit to the master branch. This accumulates to 3254 unique email addresses as of November 2020.

This data can also be used to identify where contributions to TensorFlow come from. By counting the number of unique emails for every email provider, we can get a sense of the scale of involvement of other companies, as shown in Table 4.2. A significant amount of contributions have come from other tech companies like Intel, IBM, Nvidia and Arm. The number of commits connected to each email provider shows that there are thousands of commits related to these email addresses. Although this is just the data from a single project without comparisons to other open source projects, it does show that there is substantial interest from external companies.

Email addresses and commit count are by no means flawless metrics, as they exclude people who would otherwise be considered part of the community, like those who write documentation, answer to issues or emails, and work on tasks not related to code, like management and quality assurance. Also, because of the ambiguity and different understandings of the term "community", it is hard to measure in general. Thus, these numbers should be considered a metric that tells only part of the story.

It should be fair to assume that every commit to the master branch has provided the project with some kind of value. So in terms of community members that are valuable to the project owner, commit count does function as a lower bound. As a result of the repository analysis, in May 2021, the number of people that have been

Email provider	Unique email addresses	Commit count
gmail.com	1279	9091
google.com	661	56355
github.com	476	2622
intel.com	76	1918
ibm.com	41	858
nvidia.com	35	2083
qq.com	33	102
hotmail.com	27	206
arm.com	27	671
outlook.com	24	1372
yahoo.com	22	74
163.com	16	109
microsoft.com	13	123
huawei.com	12	431

Table 4.2: The most common email addresses for commits in TensorFlow

part of the community at some point since November 2015 is *at least* 3501, of which 661 are Google employees.

Figure 4.2 shows the issue and pull request activity over time since the release in November 2015 until May 2021. In general, activity has increased since November 2015, with an all time high in March of 2019, where 1117 issues and 420 pull requests were created.

While there are a lot of open source projects that are mainly maintained by only a few very active contributors, TensorFlow is maintained by a larger team of developers. Figure 4.3 shows that the top contributor has 1347 commits to the repository, while 25 contributors have over 500 commits. This is excluding automated commits to the repository, in order to only show activity from individuals. Comparing the graph with other open source projects, the curve is considerably less steep than other OSS projects,[15] meaning that several people are sharing the majority of the workload instead of mostly being carried by only one or two.

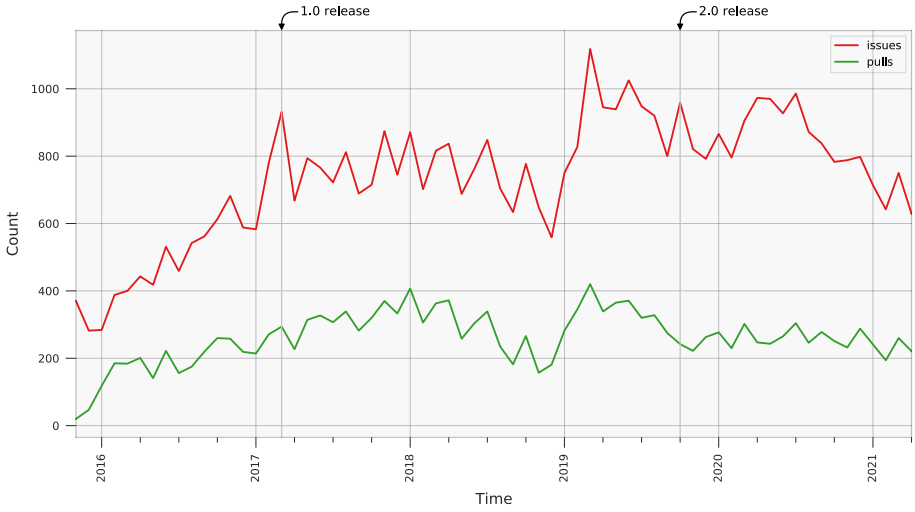

Figure 4.2: Issues and pull requests per month on the TensorFlow repository on GitHub

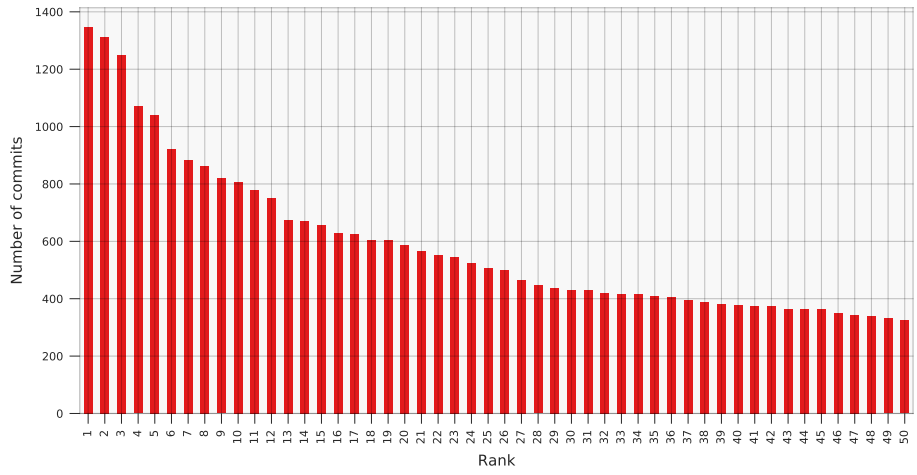

Figure 4.3: Distribution of commits among the top 50 contributors to TensorFlow

A labeling system has been created in order to categorize issues and pull requests to the respository. The types of labels range from version labels for issues related to specific TensorFlow versions, to labels for issues that are performance related. There is also a label for indicating that an issue is open to community contributors. The most used labels sorted by number of open issues are as follows[65]:

1. `type:bug` — Bug report
2. `stat:awaiting tensorflower` — Awaiting response from TensorFlow team member
3. `comp:lite` — TensorFlow Lite related issue
4. `type:feature` — Feature request
5. `comp:keras` — Keras related issue
6. `comp:ops` — Operations related issue
7. `comp:support` — Support issue
8. `TF 2.3` — Issues related to TensorFlow version 2.3
9. `type:build/install` — Build and install issues
10. `TF 2.0` — Issues related to TensorFlow version 2.0

4.2 Episodes

Following are the episodes that were examined in the case study which will shed light on parts of the underlying motivations of developers and their effects on cooperation.

4.2.1 OpenCL Support

The first issue that will be examined is the one that has received the most responses on the repository. It is also one of the first issues posted on the forum, posted on the same day as the project was made public, Nov. 9th 2015.

The issue was made as a feature request to add support for OpenCL, a noncommercial low level computation framework built on principles similar to open source software. This was proposed as an alternative to CUDA, a proprietary framework by Nvidia.[36] Support would allow TensorFlow to run on a wider range of GPUs and CPUs,[24] without relying on proprietary software.

The issue post sparked a lot of interest from the community, while some already on day one started to question Google's intentions on choosing proprietary instead of open options.

> ***Nov 9, 2015 — Developer 1***
>
> *It's strange that Google ditched open OpenCL for proprietary CUDA.*

Despite the initial choice being proprietary, Google employees supported the suggestion, and offered to help in the development of a community contribution.

> ***Jan 28, 2016 — Developer 2***
>
> *TensorFlow relies on the cuDNN library to compute convolutions on NVidia GPUs. If somebody is interested in contributing an OpenCL equivalent, we'd be happy to help.*

Such a community contribution was started by a smaller external company that collaborated with Google on an OpenCL implementation.

Sep 28, 2016 — Developer 2

We at Google have been working closely with [Developer 3] and his [Company 1] colleagues on this project for almost 12 months now. [Company 1]'s contribution to this effort has been tremendous, so we felt that we should let them take the lead when it comes down to communicating updates related to OpenCL.

Development of OpenCL support was started, but as development progressed, the community started to show their frustrations at the slow progress and the decisions made on the initial implementation.

Apr 28, 2017 — Developer 4

I find it frustrating that people are willing to write kernels in CUDA but balk at doing it for CL, even though it would reach more users and seriously grow the market ecosystem. There are direct and indirect benefits to an open platform for everyone, possibly leading to big cost savings for everyone in the long run.

Even several years after the work was started, no solution was accepted into the official TensorFlow project. Several projects had partial implementations, both inside and outside of Google.

Apr 11, 2018 — Developer 5:

It seems it's time for compressing the above discussion into one list again:

- *[Company 1] is working on a SYCL backend*
- *[Developer 6] is working on tf-coriander*
- *AMD is working on a HIP backend*
- *PlaidML only supports CPUs at the moment.*
- *Status of support for Intel GPUs is unclear.*

None of the attempts reached a finished state where the TensorFlow team was willing to make it officially part of the project. Over time, development progress and activity from the team diminished.

Sep 15, 2018 — Developer 7:

Judging by the dynamics of this bug and other forks Google has zero interest in this and will never implement this in the official repository.

By early 2019 the thread had passed 500 comments, making it hard to keep track of progress and the state of the issue. The bloated state of the thread and increased frustration from participants, lead to a maintainer closing the issue for further activity in February 2019. Since then, no activity on OpenCL support has been made from Google, and CUDA remains the only supported framework for TensorFlow computation on GPU.

4.2.2 Performance Improvement

One type of issue that demonstrates an advantage of community participation is a performance issue posted by a community member in late 2019. The issue reported that under certain conditions, build times were much longer when building with Intel MKL enabled. The issue turned out to not be related to TensorFlow itself, but rather the Intel MKL library.

Despite being posted on the TensorFlow forum, an Intel employee responded with questions on how to reproduce the issue the same day as the issue was posted. After a short conversation exchanging code snippets, it was reproduced. As the Intel employee may have suspected, the root of the problem was with Intel MKL, not TensorFlow.

Oct 9, 2019 — Developer 8:

Good suggestion. I can reproduce your performance data now. I will do some profile [sic] and let you know the result.

At this point, despite minimal participation by the TensorFlow team, the issue was being handled by qualified people. It was however a real issue that affected TensorFlow that would be beneficial to resolve for all parties.

A few months later, the Intel MKL issue was fixed by another Intel employee, lowering the build times to expected levels. The new update was soon applied to TensorFlow, and the user that submitted the issue was able to confirm that the issue

was fixed.

> ***Oct 19, 2019 — Developer 9:***
>
> *Thanks for all the information. Looking into this, I will update once I find the root cause.*
>
> ***Jun 23, 2020 — Developer 10:***
>
> *The current performance is better now. A fix was added with DNN 1.2*

The issue was closed around 8 months after it was opened. In a conventional software environment, such an issue may have been much harder to resolve. It may have been more difficult for the user to identify the problem if the project architecture was concealed. The Intel employee would not have been able to discover the issue as quickly, and communication would be harder between the several parties involved.

4.2.3 Implement Fast Fourier Transform Ops

In December 2015, an user posted a feature request to the GitHub issue board requesting support for fast Fourier transforms (FFT) as a TensorFlow Operation (commonly referred to as an "op"). The user posted because of a limitation with the framework they had encountered during use. The user quickly received a response from a TensorFlow team member, and a technical discussion about requirements and implementation was started. A need for several different versions of FFT was needed, with support for running on both GPU and CPU.

After just a couple of weeks a partial solution with some FFT methods running on the GPU was implemented by the TensorFlow team, and merged into the project. After this, progress on the feature slowed down somewhat, and an effort to engage external contributors was started. A TensorFlow team member commented that the team was open to contributions from the community, and the "stat:contrubutions welcome" label was added to the issue.

By the second half of 2016 a Google employee took responsibility and started work on the remaining parts of the fast Fourier transform op. In March 2017, the remaining FFT versions were completed for the GPU, leaving just the CPU implementation. At

this point, an external developer joined the conversation with a solution to most FFT computations on CPU.

> ***Mar 26, 2017 — Developer 11:***
>
> *[Developer 12], I have working CPU/GPU kernels for complex FFTs based on Eigen but have some issues integrating your recent RFFT changes. Would you be up for a chat to iron out those bumps so we can add the CPU kernels?*

> ***Mar 27, 2017 — Developer 12:***
>
> *happy to chat! [...] Want to email me and we'll go from there? Also, are the changes pushed publicly anywhere?*

After about a month the feature was accepted into TensorFlow.

> ***May 18, 2017 — Developer 12:***
>
> *complex FFT and IFFT as well as the RFFT are now in master. The IRFFT still needs a bit of work.*

The final piece of the puzzle was also solved by an external contributor. Within a few days the final part of the feature was added by a first-time contributor, which was accepted and merged promptly.

> ***May 23, 2017 — Developer 13:***
>
> *[Developer 11] Very much thanks for your contribution! I've made up the IRFFT part*

> ***May 24, 2017 — Developer 12:***
>
> *Thanks to [Developer 11] and [Developer 13], I think we can call this done!*

After this, the issue was closed, around one year after it was submitted.

4.2.4 ONNX Support

In September 2017 Facebook and Microsoft released a deep learning model format called Open Neural Network Exchange (ONNX). The format was intended to be a standard for ML models that would enable interoperability between frameworks by enabling the transfer of model data across frameworks.[9]

The same day as ONNX was announced, an issue was posted on the TensorFlow GitHub repository asking for ONNX support, which received a lot of support from the community, but much less from the TensorFlow team. The ONNX team proceeded with development and submitted a pull request with the requested feature. The feature was not accepted due to an unwanted dependency.

> ***Nov 17, 2017 — Developer 14:***
>
> *We're happy to consider something like this as a separate repository in github.com/tensorflow, but would prefer to avoid moving it to contrib due to the onnx dependency. Perhaps let's discuss on the issue?*

When the 1.0 version of ONNX was released in December 2017, there was support from every popular ML frameworks except TensorFlow.

> *Support for ONNX is available now in many top frameworks and runtimes including Caffe2, Microsoft's Cognitive Toolkit, Apache MXNet, PyTorch and NVIDIA's TensorRT. We also have community contributed converters for other projects such as TensorFlow.*[54]

As activity on the TensorFlow repository shows, this was not due to lack of effort from the ONNX teams' side, but rather a lack of cooperation from the TensorFlow team.

> ***Mar 29, 2018 — Developer 15:***
>
> *As much as I don't understand the fear of ONNX dependency, I'm willing to continue working on another pull request that isolates the bad part of the onnx dependency and only retain the part relevant to Tensorflow conversion.*

At this point TensorFlow team members stopped participating in the discussion

despite it becoming one of the most popular feature requests for the project. As time passed without any response from the TensorFlow team, questions about the teams agenda was questioned by the community:

> ***Mar 29, 2018 — Developer 16:***
>
> *I know this issue might not get that much attention from Google since they have their own interests, but for the AI research and development community having support for a standard format that is portable across frameworks and runtimes (like TensorRT) is HUGE [...]*
>
> ***Jun 3, 2018 — Developer 17:***
>
> *So what is the issue here? Does TF prefer to move ONNX to another repo in order to prevent better interoperability?*

The "community contributed converters" mentioned above remains the solution for conversions between TensorFlow and ONNX, but they are not officially supported by Google or TensorFlow and the source code resides outside of the TensorFlow organization.

Chapter 5

Discussion

Based on the background knowledge gathered from the literature study, a few central themes were derived from the findings of the case study. Table 5.1 shows how these results were deduced from the episodes.

The research questions outlined in the introduction are a result of these themes, and the topic of this chapter. They will be discussed one by one in light of the case study findings.

1. (Section 5.1) What kinds of motivational differences appear in commercial open source projects?

2. (Section 5.2) How do diverse motivations have an effect on cooperation?

Source Data	Derived Theme	Central Theme
"Does TF prefer to move ONNX to another repo in order to prevent better interoperability?"	Wanting to maintain users despite community wish to be interoperable reveals that there is a divide between the two parties.	Conflicting values and motivations creates a divide with tension and degraded trust between company and community.
"I know this issue might not get that much attention from Google since they have their own interests"		
"It's strange that Google ditched open OpenCL for proprietary CUDA."	Different prioritisation because of different value assessment. Using libre software (hacker ethics) vs. monetary return on investment (commercialism) creates tension.	
"I find it frustrating that people are willing to write kernels in CUDA but balk at doing it for CL."		
"I have working CPU/GPU kernels for complex FFTs [...]. Would you be up for a chat to iron out those bumps so we can add the CPU kernels?"	Cooperation on the right issues is mutually beneficial. Intrinsically motivated volunteers are willing contribute feature gifts that supplement employees work.	Within boundaries, diverse motivations and goals enables effective collaboration between contributors that lead to mutually beneficial contributions.
"Thanks to [Developer 11] and [Developer 13], I think we can call this done!"		
"Good suggestion. I can reproduce your performance data now. I will do some profile and let you know the result."	Open discourse and diverse motivations enables all parties (volunteers, employees and external companies) to collaborate effectively, and achieve goals that are in everyones best interest.	
"Thanks for all the information. Looking into this, I will update once I find the root cause."		

Table 5.1: Central themes derived from the case study

5.1 Motivational Differences

Though there has been a substantial amount of research on the motivations of both individuals and organizations within open source development, not much effort has been made to emphasize the differences between the two parties. Both previous literature and the case study shows that there is a significant gap between the motivations of the individuals that spend their free time on OSS development and the profit-driven organizations that use open source development practices in their work. The case study further suggests that these have a real impact on open source software development.

Looking first at commercial motivations, a lot of similarities was found between the case and literature study, but the case also gives new insights into how commercial motivations become visible in practice.

The literature study points to one foundational motivator by commercial actors in open source: the economic benefits. But OSS is an interesting topic in part because of the absence of monetary benefits, or rather that the economic benefits are ambiguous and appear as a subsequent effect of a successful open source software project instead of direct licensing fees.

In order for the projects to be successful, commercial actors need to adhere to certain norms to maintain trust from the open source community. This effect does part of the work of bridging a motivational gap between company and volunteers. This way, the organization's activities become more similar to noncommercial actors even though their goals can be very different. However, the case study reveals several cases where the motivational differences become visible despite this effect.

These visible differences are similar to what was found in the literature study. As suggested by Bonaccorsi and Rossi [8], Google's motivations are economic, social and technological. Google is interested in maintaining a strong community around the project and expanding the technology capabilities of it, so that it is useful and provides value to themselves and others. Employed maintainers spend time and effort on interacting with the community, and sometimes implement entirely new features on requests from users. These are activities very similar to those of altruistically motivated individuals, but by a profit-driven company.

Though it is not explicitly stated anywhere, Google can be assumed to be guided by the intention of turning a profit off TensorFlow, but the economic incentives are more ambiguous than many other commercial OSS projects. There is no dual-licensing, support services, and related services are few and rarely promoted.

In the case study, two concrete examples came up that show how motivations and values are misaligned. Firstly in Section 4.2.1, Google have chosen to rely on proprietary software despite an open source alternative being available. This sort of dependency is what Stallman tries to avoid with the GNU project, and further goes against the Hands-On Imperative and access to information from Levy's hacker

ethics.[28]

The second example in Section 4.2.4 suggests that the company discouraged interoperability with other competing frameworks, presumably to retain users. This is again in contrast to one of the points from hacker ethics, *"All information should be free"*, and in conflict with a potential lock-in strategy where program data is intentionally kept within the framework.

The mostly economical motivations of companies are in stark contrast to the volunteers that take part in the project. Though financial incentives in some cases can be part of the reason why some of the participants contribute, volunteers are shown to be more motivated by social incentives like reputation and community affiliation, as well as technical experience and fun. The case shows that volunteer's beliefs are still similar to hacker ethics, where contributing is seen as an act for the "greater good".

The study of TensorFlow has shown examples of how values and motivations differ between company and volunteer, as well as how it can influence cooperation when solving issues or adding features to the framework. In practice, however, these are outnumbered by the issues where motivations are not an issue, and collaboration goes on without having to question each other's intentions.

For individuals, most of the incentives do not change from commercial and non-commercial projects. There are still reputational benefits, a community to partake in and technical challenges to learn from. The difference for the volunteer is that the owner of the project has a set of goals and values that the individual may not subscribe to. A volunteer may choose to not partake because of these differences, but clearly lots of volunteers do non the less.

In general, everyone involved is interested in improving the product's use value. New features, performance improvements, upgrades and bug fixes are usually mutually beneficial, which is why conflicts are rare. There is plenty of work to be done within the scope of the framework that benefits both sides, which is what is discussed in the majority of issues. In these cases volunteers get intrinsic and community benefits while companies get valuable community contributions.

5.2 Effects on Collaboration

This study has shown both in literature and practice that motivations of participants in commercial open source aren't always homogeneous. Further, the project shows that commercial projects don't function as a collective when conflicting interest appears in development.

The examples show that part of the heuristics of Bonaccorsi and Rossi [8], where companies are *"Conforming to the values of the Open Source community"*, has its limitations, and that Google are willing to break with the social norms of OSS in favor of economics.

As described in Section 4.2.4, Google was not interested in accepting features that increased the project's interoperability. Google's lack of cooperation sent signals that the project isn't guided by the interests of the community as much as the company. The most popular feature suggestions on the project were abandoned, even though a completed implementation was proposed.

Sending these kinds of signals damages the trust between contributors and owners. As explained by Osterloh et al. [41], if a contributor gets the impression that the company is breaking the unwritten social contract, volunteers will be inclined to leave. In addition to the blocked feature, this could limit the potential for future cooperation and innovation.

From the perspective of users and the community, these differences become constraints on the project progression compared to noncommercial open source projects where motivations are more uniform. Limiting the kinds of features users of the software would want, could make an open source tool more similar to a proprietary one, where development is driven exclusively by the company and its employees. Effects like lock-in and the usage of proprietary dependencies are something that are rarely seen in libre software, and show part of how commercial open source is different.

This suggests that TensorFlow is not always a "best of both worlds" situation as proposed by Hippel and Krogh [23], rather OSS with an asterisk. This is because TensorFlow is primarily released as a strategic Google product, and less as a donation to the field of machine learning. Although Google chooses to follow OSS development

practices, uses an open source license, and encourages community activity, this is not an end goal itself. Google isn't going to make decisions that don't make sense monetarily.

This diversity of values and motivations is one way in which commercial open source projects are distinct. Any team can encounter conflicts, but when there are clearly set goals and core values that every member can subscribe to, there is a shared purpose to refer to when conflicts arise. When there are differences in the foundational values of participants in commercial open source, there is no common ground when core values become relevant to the discussion.

The episodes in this study indicate that commercial open source projects have a flaw in its community. But even though these differences have sparked conflict, it is apparent that the participants have been able to work together constructively on the macro scale. Despite their differences, the two parties collaborate on the same pieces of software with the shared goal of improving and maintaining it. The activity and scale of OSS today is in itself proof that this form of collaboration is able to support the development of vast amounts of software.

Another way of framing the situation is to consider the open source project the enabler of collaboration instead of the conflicting goals as limitations.

Commercial open source projects like TensorFlow creates a situation where individuals and organizations interact on the same platform, doing similar work, because of two fundamentally different reasons. They are communities of practice, but far from homogeneous regarding motivations. The software serves as a shared field of interest for this diverse set of people. Commercial open source projects like TensorFlow functions as boundary objects[59] between developer groups with different motivations.

In line with Bechky[5], the commercial open source project is the common ground that enables differently motivated participants to work together on the same issues. The community of open source volunteers seems to have acceptance of commercial motivations, at the same time as companies continue to collaborate with open source developers that don't have the same monetary priorities. In the grand scheme of the project, the few disagreements may be negligible in an otherwise well functioning synergy.

In Section 4.2.2, employees of external companies are incentivized to fix problems with their project they were responsible for. Their role in the project supplements the other participants by focussing on the issues that concerns them, and not so much TensorFlow team members. The issue affected the project as a whole, and was in everyone's interest to resolve, but the motivations of these participants solved the issue without much effort from Google.

When both parties are interested in achieving the same objectives, different underlying values are not a hindrance to cooperation. Cooperation on features that appeals to both parties interests are possible and beneficial to both. Making a well functioning and useful tool are part of this shared motivation between company and volunteers.

Chapter 6

Conclusion

The open source movement has initiated a radical shift in software development in the twentieth century. Organizations have opened up their in house projects to the world and encouraged its users to partake in their development. This has essentially removed the distance from producer to consumer, and made every step and decision entirely visible to the public, as well as open for critique.

Despite its effort to steer away from appearing this way, commercial open source software is at its core still commercial. This may be contrary to its appearance to outside viewers, as the business model of commercial open source projects can be ambiguous when there is no direct monetary benefit for the owner organization. Most open source software projects appear open, free, collaborative and customizable with no strings attached. In addition, by partaking in open source development, these companies are expected to adhere to the same norms that activists pioneered around the turn of the century.

This can initially give the impression that companies and volunteers have the same goals, while in reality their beliefs are fundamentally different. The ethics of volunteers are still similar to hacker ethics of the 60s, especially in terms of freedom of information. Commercial strategies can in some cases be in conflict with these ethics, which means the company has to choose between revenue and the trust of its

community.

Despite this clear divide in most commercial open source projects, little empirical data has been collected on its effect on collaboration. What this study show is that there are practical examples of cases where motivational and value differences lead to conflicts within open source software development. The findings suggest that a commercial owner will indeed put its own interests first, steering away from the idea that choosing the open source model is purely a "charitable" gesture.

Since conflicts can appear when a feature interferes with one of the parties intentions, being aware of these differences can be important to understand for participants in OSS projects as well as the users that rely on it. The capabilities of commercial open source tools show that innovation within these boundaries certainly are possible, though maybe not without limitations.

Yet, commercial open source has brought companies, researchers, students and hobbyists together to create a shared product that is valuable to a very large number of users outside of the participant group. The radically open development model has created a community where thousands of developers work on software that makes its way into every part of our society.

6.1 Limitations of the Study

This study is certainly not without its limitations, and there are thousands of different examples of commercial open source practices available that this study doesn't have the time to cover. This is just one of many possible perspectives in a complex issue, but one that shows that there is plenty to learn on this topic going forward. The data the study was based on, is also just a small selection of what was available from the forum, which may be far from representative of most developer interactions.

Overall, more quantity is needed for greater confidence. If there are applications of these findings, a sample size greater than one would probably be appropriate.

Motivations are hard to reliably identify on technical forums, which leads to assumptions and interpretations that may not be accurate. Performing interviews or surveys with developers could have been advantageous to increase the credibility of

the results.

6.2 Further Work

It's likely that the topic of motivations in commercial open source will become more and more important as commercial open source software becomes a larger part of digital infrastructure. Studies on whether potential feature limitations in commercial open source projects also create limitations of the software that rely on it could be important for the developers that are making technology choices.

Studies on how conflicts in motivation affect how volunteers contribute in commercial open source projects. If conflicts are driving participants or users away from the project, companies may want to change the way they interact with users, and communicate their goals.

The results of the study indicate that motivations may function as delegators of tasks to contributors with matching motivations. Further studies on the type of tasks that are prioritized by different groups could be very useful to maintainers.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Apple Inc. (2020). FoundationDB - the open source, distributed, transactional key-value store. [Online; accessed 5. Sep. 2020].

[3] Babel (2021). Babel Contributors. [Online; accessed 29. May 2021].

[4] Banton, C. (2020). Loss Leader Strategy. *Investopedia*. [Online; accessed 21. Sep. 2020].

[5] Bechky, B. A. (2003). Sharing meaning across occupational communities: The transformation of understanding on a production floor. *Organization science (Providence, R.I.)*, 14(3):312–330.

[6] Black Duck Open Hub (2020). Compare Repositories - Open Hub. [Online; accessed 9. Oct. 2020].

[7] Bogensberger, B. (2019). npm Blog Archive: Managing JavaScript in the Enterprise. [Online; accessed 13. May 2021].

[8] Bonaccorsi, A. and Rossi, C. (2006). Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business. *Knowledge, Technology and Policy*, 18(4):40–64.

[9] Candela, J. Q. (2017). Facebook and Microsoft introduce new open ecosystem for interchangeable AI frameworks - Facebook Research. [Online; accessed 25. Mar. 2021].

[10] Chacon, S. (2014). Pro git.

[11] Clojure (2021). Clojure Contributors. [Online; accessed 29. May 2021].

[12] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In *NIPS*.

[13] Dougherty, C. (2015). Astro Teller, Google's 'Captain of Moonshots,' on Making Profits at Google X. *Bits Blog*. [Online; accessed 13. Nov. 2020].

[14] Eghbal, N. (2016). *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure*. Ford Foundation.

[15] Eghbal, N. (2020). *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press.

[16] Elasticsearch B.V. (2021). Elasticsearch. [Online; accessed 14. Apr. 2021].

[17] Fortune (2021). Global 500 Technology. *Fortune*. [Online; accessed 31. May. 2021].

[18] Google (2020). Open source by the numbers at Google. [Online; accessed 26. May 2021].

[19] Google (2021). Google Open Source - opensource.google. [Online; accessed 26. May 2021].

[20] Hacker News (2015). Hacker News Discussion | TensorFlow: open-source library for machine intelligence. [Online; accessed 8. Nov. 2020].

[21] Hall, M. (2006). The End of Unix? *Computerworld*. [Online; accessed 21. Oct. 2020].

[22] Halvorsen, H. B. (2020). Refining Commercial Open Source: Driving Adaption and Growing Ecosystems. Master's thesis, NTNU.

[23] Hippel, E. v. and Krogh, G. v. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. *Organization science (Providence, R.I.)*, 14(2):209–223.

[24] Khronos Group (2021). OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems. [Online; accessed 24. May 2021].

[25] Krogh, G. v., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research policy*, 32(7):1217–1241.

[26] Lave, J. (1991). Situated learning : legitimate peripheral participation.

[27] Lévénez, É. (2020). Unix History. [Online; accessed 5. Oct. 2020].

[28] Levy, S. (1984). *Hackers: Heroes of the computer revolution*, volume 14. Anchor Press/Doubleday Garden City, NY.

[29] Linux (2021). Linux Contributors. [Online; accessed 29. May 2021].

[30] Markoff, J. (2012). In a Big Network of Computers, Evidence of Machine Learning. *N.Y. Times*. [Online; accessed 12. Nov. 2020].

[31] Markus, M. L. and Agres, B. M. C. E. (2000). What makes a virtual organization work? *MIT Sloan management review*, 42(1):13.

[32] MongoDB, Inc. (2021). MongoDB. [Online; accessed 14. Apr. 2021].

[33] Nadella, S. (2018). Microsoft + GitHub = Empowering Developers - The Official Microsoft Blog. [Online; accessed 5. Sep. 2020].

[34] Netscape Communications Corporation (1998). Netscape announces plans to make next-generation communicator source code available free on the net [archived, october 1, 2002]. [Online; accessed 9. Oct. 2020].

[35] Node.js (2021). Node.js Contributors. [Online; accessed 29. May 2021].

[36] Nvidia (2021). CUDA Zone. [Online; accessed 24. May 2021].

[37] Open Source Initiative (2002). History of the OSI [archived, october 1, 2002]. [Online; accessed 9. Oct. 2020].

[38] Open Source Initiative (2020a). About the Open Source Initiative. [Online; accessed 19. Oct. 2020].

[39] Open Source Initiative (2020b). The Open Source Definition. [Online; accessed 19. Oct. 2020].

[40] Open Source Initiative (2021). Licenses & Standards | Open Source Initiative. [Online; accessed 2. Apr. 2021].

[41] Osterloh, M., Rota, S., and Wartburg, M. v. (2001). Open source - new rules in software development. *Institute for Research in Business Administration. Uni Zürich.*

[42] Oxford University Press (2021). commercial, adjective. In *Oxford Learner's Dictionaries*. Oxford University Press. [Online; accessed 10. Mar. 2021].

[43] Page, L. (2015). G is for Google. *Google*. [Online; accessed 5. Apr. 2021].

[44] PyTorch (2021). PyTorch GitHub repository. [Online; accessed 1. May. 2021].

[45] Raymond, E. S. (1998). Homesteading the noosphere. *First Monday*, 3(10).

[46] Raymond, E. S. (2001). The cathedral and the bazaar : musings on linux and open source by an accidental revolutionary.

[47] Raymond, E. S. (2004). The art of unix programming.

[48] Reiss, S. (2012). Intrinsic and extrinsic motivation. *Teaching of Psychology*, 39(2):152–156.

[49] Roberts, J. A., Hann, I.-H., and Slaughter, S. A. (2006). Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Management science*, 52(7):984–999.

[50] Royce, W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on software engineering*, ICSE '87, pages 328–338. IEEE Computer Society Press.

[51] Rust (2021). Rust Contributors. [Online; accessed 29. May 2021].

[52] Røsvik, J. T. (2020). Open vs. Closed Code: Understanding Commercial Motivations. Project thesis, NTNU.

[53] Sandjideh, A. M. (2017). Announcing TensorFlow 1.0. [Online; accessed 12. Nov. 2020].

[54] Sarah Bird, D. D. (2017). ONNX V1 released - Facebook Research. [Online; accessed 25. Mar. 2021].

[55] Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management science*, 52(7):1000–1014.

[56] Stallman, R. (1983). Initial announcement. [Online; accessed 7. Oct. 2020].

[57] Stallman, R. (1989). Gnu general public license, version 1. [Online; accessed 7. Oct. 2020].

[58] Stallman, R. (2020). Why Open Source misses the point of Free Software. [Online; accessed 19. Oct. 2020].

[59] Star, Susan Leigh, J. R. G. (1989). Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, 19(3):387–420.

[60] TensorFlow (2015). Home — TensorFlow. [Online; accessed 30. Nov. 2020].

[61] TensorFlow (2020a). Forums | TensorFlow. [Online; accessed 5. Nov. 2020].

[62] TensorFlow (2020b). TensorFlow Homepage. [Online; accessed 8. Dec. 2020].

[63] TensorFlow (2020c). Tensorflow: Initial commit of tensorflow library. [Online; accessed 4. Nov. 2020].

[64] TensorFlow (2021a). TensorFlow GitHub repository. [Online; accessed 1. May. 2021].

[65] TensorFlow (2021b). TensorFlow Repository Labels. [Online; accessed 25. Jan. 2021].

[66] TensorFlow Team (2019). TensorFlow 2.0 is now available! [Online; accessed 12. Nov. 2020].

[67] The Free Software Foundation (2020). The Free Software Foundation (FSF) is a nonprofit with a worldwide mission to promote computer user freedom. We defend the rights of all software users. [Online; accessed 8. Oct. 2020].

[68] The Free Software Foundation (2021). What is free software? [Online; accessed 29. May 2021].

[69] The Internet Archive (2021). Wayback Machine. [Online; accessed 6. Apr. 2021].

[70] Thibodeau, P. (2013). As Unix fades away from data centers, it's unclear what's next. *Computerworld*. [Online; accessed 21. Oct. 2020].

[71] Torvalds, L. (1991). What would you like to see most in minix? [Online; accessed 8. Oct. 2020].

[72] Torvalds, L. (2001). Just for fun: the story of an accidental revolutionary.

[73] Tozzi, C. (2017). *For Fun and Profit: A History of the Free and Open Source Software Revolution.* History of Computing. MIT Press, Cambridge.

[74] Wenger, E. (1998). Communities of practice : learning, meaning, and identity.

[75] Wenger-Trayner, E. and Wenger-Trayner, B. (2015). Introduction to communities of practice | Wenger-Trayner. [Online; accessed 18. Feb. 2021].

[76] Young, R. (1999). *Giving it Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry*, chapter 9, pages 56–61. O'Reilly.

NTNU
Norwegian University of
Science and Technology