Impactful contributions of usability practitioners to open source software projects: a multiple case study

University of Oulu
Faculty of Information Technology and Electrical Engineering / Degree Programme of Information Processing Science
Master's Thesis
Janne Niemelä
1.6.2021

Abstract

Open source software (OSS) has been described as being designed by and for technically advanced users. As OSS has been gaining popularity among non-technical users, concern about its usability has been raised, as it is difficult for technically-minded developers to design for average users. Hiring usability experts to represent the needs of average users has been used in commercial software development as an effective solution for improving usability. It has been also suggested as a way of addressing the usability issues of OSS, but it has been observed that it is often difficult for usability experts to contribute to OSS so that their work has a major impact on the usability of the software.

In this thesis, a multiple case study of four usability interventions was conducted. The cases were a part of a larger research program called UKKOSS, which aims to test ways how usability experts can meaningfully contribute to OSS by conducting usability interventions, where student teams act as usability practitioners who enter OSS projects and carry out usability work on them. This study examined how OSS developers reacted to four of those usability interventions by examining the data gathered during those interventions. The analysed data included documents, such as summary reports, communication logs, project plans, and reports on the conducted usability activities. The larger goal of studying these cases was to gather information on how usability practitioners can conduct impactful usability work on OSS projects. The outcomes of the cases were examined through the lens of prior research, and the factors that may have contributed to the success of the cases were examined through cross-case analysis.

The developers welcomed the usability work of the usability teams in generally all of the four cases, but the actual impacts the interventions had varied from none of the suggested usability changes being implemented to most of them being implemented to the software. The outcomes of the most successful cases suggest that an approach where usability practitioners implement their suggested changes themselves after discussing about them with the core developers, establishing trust with the developers by contacting them via voice call or video conferencing instead of using only asynchronous communication, and making usability reports as persuasive as possible by including user testing metrics which strengthen the validity of the issues, should be studied further to evaluate if they can have a positive effect on the impact of the work of usability practitioners. The main contributions of this research were supporting the prior research on the obstacles faced by usability experts entering OSS projects by supporting it with empirical evidence and proposing new areas of research on the subject based on the outcomes of the cases.

Keywords
Open source software, usability, UKKOSS, OSS

Supervisor
PhD, university lecturer, Mikko Rajanen

Foreword

I would like to thank my supervisor Dr. Mikko Rajanen for guiding my work and managing the UKKOSS research programme of which usability intervention cases were examined in this thesis, thus making this research possible. I would like to also thank all the other students who participated in the UKKOSS projects 14-17 and gathered valuable research data which was analysed in this thesis, and Dr. Raija Halonen for teaching me the basics of conducting scientific research.

Janne Niemelä

Oulu, June 1, 2021

Contents

Abstract 2
Foreword 3
Contents 4
1. Introduction 5
2. Prior research 7
    2.1 Open source software 7
        2.1.1 The history of open source software 8
        2.1.2 OSS development process 10
        2.1.3 Motivations for participation in OSS development 11
        2.1.4 The hierarchical structure and decision-making in OSS projects 12
    2.2 Usability 15
        2.2.1 User/human-centered design 16
        2.2.2 Usability engineering methods 17
    2.3 Usability in OSS projects 19
        2.3.1 Usability issues in OSS 19
        2.3.2 Usability practitioners' barriers to contributing in OSS projects 21
        2.3.3 UKKOSS research programme 25
3. Research methods 29
    3.1 UKKOSS research programme as the basis of research 29
    3.2 Research question 29
    3.3 Case study process 29
        3.3.1 Design and planning 29
        3.3.2 Collecting data 31
        3.3.3 Analysis 32
        3.3.4 Reporting 33
4. UKKOSS usability intervention cases 34
    4.1 Case 1: UKKOSS 14 (Mumble) 34
    4.2 Case 2: UKKOSS 15 (Task Coach) 34
    4.3 Case 3: UKKOSS 16 (HandBrake) 35
    4.4 Case 4: UKKOSS 17 (Streama) 36
    4.5 Summary 36
5. Findings 39
    5.1 How did the open source software communities react to usability improvement activities conducted by external usability practitioners? 39
        5.1.1 Reactions to UKKOSS 14 (Mumble) 39
        5.1.2 Reactions to UKKOSS 15 (Task Coach) 41
        5.1.3 Reactions to UKKOSS 16 (HandBrake) 42
        5.1.4 Reactions to UKKOSS 17 (Streama) 44
        5.1.5 Summary 47
6. Discussion & implications 49
    6.1 Critique of this research 52
7. Conclusions 53
    7.1 Limitations of this study 53
    7.2 Possible future research 54
References 55

1. Introduction

It has been argued that open source software often has usability issues (Andreasen et al., 2006; Feller & Fitzgerald, 2000; Lisowska Masson et al., 2017; Nichols & Twidale, 2003). OSS projects are often started due to personal need of the developers (Moody, 2001, as cited in Crowston et al., 2004; Raymond, 1999; Vixie, 1999, as cited in Crowston et al., 2004), and the developers of the software are often its users (Andreasen et al., 2006; Crowston et al., 2004; Nichols & Twidale, 2003). The philosophy of user-centered design emphasises that it is hard for software developers to design for non-technical users (Nichols & Twidale, 2003). As the user base of OSS grows to include more and more non-technical users, there is a need for usability considerations addressing non-technical users' perspective (Feller & Fitzgerald, 2000; Nichols & Twidale, 2003; Rajanen & Iivari, 2019). The lack of usability experts in OSS has been identified as a problem for the usability of OSS, and it has been suggested that they could bridge the gap between the developers and the average users (Nichols & Twidale, 2003). In commercial software development, usability experts typically have the authority needed to represent average users (Nichols & Twidale, 2003). In OSS development, they often face difficulties when attempting to influence the design of the software (Andreasen et al. 2006; Çetin et al., 2007; Nichols & Twidale, 2003; Rajanen & Iivari, 2015; Rajanen et al., 2011; Rajanen et al., 2015). Various usability interventions by usability teams to OSS projects have been studied in UKKOSS research programme, which aims to find and test effective ways of introducing usability activities to OSS projects by utilising student usability teams guided by researchers (Rajanen & Iivari, 2019). Further research on how usability practitioners can effectively contribute to OSS has been proposed (Rajanen et al., 2012).

The main goal of this thesis is to investigate how usability practitioners can participate in OSS development in a way that their work can have an impact on the software's usability. The research question of this study is: *"How did the open source software communities react to usability improvement activities conducted by external usability practitioners?"* The research method is multiple case study of four usability interventions to OSS projects that were conducted from 2015 to 2016 as a part of UKKOSS research programme. The author participated in one of the cases. The reactions of the OSS developers are described though detailed case descriptions, the outcomes of the cases are reflected on prior research, and cross-case analysis is conducted in order to identify possible factors that may have had an effect on the outcomes of the cases. The main contributions of this research are supporting empirically the prior research on the issues usability practitioners encounter when contributing to OSS and proposing new areas of research based on the results of this study for gaining further insights on impactful usability work.

This thesis is split into 7 chapters, Introduction, Prior research, Research methods, UKKOSS usability intervention cases, Findings, Discussion & implications, and Conclusions. Prior research discusses the existing literature on open source software, usability, and usability of open source software. It covers themes like the common definitions of OSS and usability, the history of OSS and usability research, the OSS process, decision-making in OSS, user-centered design, usability engineering methods, usability problems in OSS, the issues usability practitioners encounter when participating in OSS development, and the UKKOSS research programme. Research methods covers the used research method and data gathering. It describes the use of case study as the research method, presents the research question, and explains the context of the analysed cases as a part of a research programme. UKKOSS usability intervention cases chapter describes the cases in detail, and in the Findings chapter, the research question is

answered by describing how the OSS communities reacted to the usability intervention. The Discussion & implications chapter analyses outcomes of the case studies in the context of prior research. In the final chapter Conclusions, the results of this study are summed up, and the limitations of this study and possible future research suggestions are discussed.

2. Prior research

This chapter discusses the prior research on open source software, usability, and usability in open source software. The open source software subchapter discusses the definitions of open source software, its history, OSS development process, motivations for participating in OSS development, and decision-making in OSS projects. The usability subchapter discusses the definitions of usability, the history of usability, user-centered design, and usability engineering methods. The usability in OSS projects subchapter discusses the state of usability in OSS projects, examines what kinds of problems usability practitioners encounter when working on OSS projects, and describes the UKKOSS usability intervention research programme.

2.1 Open source software

Open source software is software that is licenced in a way that allows the users to access and modify the source code behind its pre-compiled binary (Bretthauer, 2002). There are many kinds of open source licenses with different rights and restrictions. For example, some of them require that derivative software has to use the same license, like the GNU General Public License (GPL). (Gacek & Arief, 2004.) The GNU acronym stands for Gnu's Not Unix (GNU, 2015). OSS development projects are usually Internet-based communities of software developers (von Krogh & von Hippel, 2003). The scale of OSS projects can vary from one developer coding the software for personal use to huge projects with hundreds of developers, such as Linux, Firefox, LibreOffice and Blender (Rajanen & Iivari, 2019). The term OSS should not be mixed with shareware or public domain software, which refer to software that is free to use but usually do not allow the user to access the underlying source code. The concept of open source does not disallow monetising the software or its related services. Open source software can be monetised by for example charging for its packaging, support or distribution. (Bretthauer, 2002.) While most of OSS is built purely voluntarily, some organisations also sponsor and pay developers so they can focus on working on OSS full-time (Hertel et al., 2003).

Gacek and Arief (2004) argue that although open source projects' characteristics can vary greatly between projects, there are two constants that are common in all of them: they are always developed by the users, and they follow the criteria of the Open Source Definition. Open Source Definition is a set of criteria defined by Open Source Initiative (OSI), an educational and advocacy organisation for the open source phenomenon (Open Source Initiative, 2018), that define the distribution terms of open source software. They include rules such as these:

- The license must allow free redistribution of the software as a part of a software collection that contains software from multiple different sources
- The software has to include source code, and distribution must be allowed in both source code and compiled form
- Derived versions of the software should be distributed using the same license as the original software
- The distribution of modified versions of the software can be disallowed only if distributing patch files for modifying the software is allowed
- The license cannot discriminate against persons or groups, or disallow using the software for business purposes or in a specific domain

- The rights of the license must be automatically passed to everyone to whom the software is redistributed
- The license cannot demand restrictions on other software, and the software cannot be restricted to a specific technology

These terms were originally adapted from Debian Free Software Guidelines. (Open Source Initiative, 2007.) Open source licenses are licenses that follow these rules. In order to be approved by OSI, a license must pass their license review process. (Open Source Initiative, n.d..)

2.1.1 The history of open source software

The roots of open source software can be traced back to the 1950's and 1960's, when software was bundled with its hardware, and the users were free to exchange modified source code among themselves (Hars & Ou, 2002). Most of the software in the early days of programming was developed in academic and corporate laboratories by engineers and scientists. Exchanging software freely to build upon each other's software was considered a normal work practice. (von Krogh & von Hippel, 2003.) Since then, software begun moving towards commercialisation, resulting in software becoming increasingly proprietary (Hars & Ou, 2002).

In the 1980's, Massachusetts Institute of Technology (MIT) decided to license some of the code created by the employees at the MIT's Artificial Intelligence Laboratory to a commercial company. The company restricted the access to the software's source code to only the employees, preventing non-involved people, such as people who had worked on it and left MIT, using it. As a counterreaction to this and the general trend towards proprietary software protected by copyright licenses, Richard Stallman, a researcher working at the Artificial Intelligence Laboratory, founded Free Software Foundation (FSF). (von Krogh & von Hippel, 2003.) FSF is an organisation raising funds for promoting the freedom to modify and share software (Bretthauer, 2002). Its core ideas can be credited for being the conceptual foundation of modern open source software, but they have been often critiqued for being too ideological (Hars & Ou, 2002). In 1985, Stallman wrote the GNU manifesto, a document where he described his ideological stance on software freedom. He believed that programmers should share the source code of their software to the users and considered it wrong to hoard information instead of sharing it. He called for other developers to help him build an open source operating system, GNU. (GNU, 2015.) The first open source license called GPL (GNU General Public License) was created for this project. It ensures unrestricted access to the source code of the software and its derivatives. Derivative software is also forced to have the same license as the parent software. This licensing style is called copyleft. (Fitzgerald, 2006.) Stallman and his new community of developers started working on GNU. GNU was based on UNIX, an open source operating system developed by Ken Thompson, which first version was released in the late 1960's. During the GNU project, many kinds of tools and utilities were produced that Linus Torvalds, the lead developer of the Linux operating system's kernel, used in the development of Linux in the early 1990's. (Hars & Ou, 2002.) Linux was distributed under the GPL license (Fitzgerald, 2006). Modern Linux can be described as a combination of Linux kernel, GNU software, and other supplementary software parts (Hars & Ou, 2002).

In the late 1990's Eric Raymond, a programmer and at the time the co-founder of a small free-access internet service provider called Chester County Interlink, was inspired by the success of the Linux project and decided to figure out why the distributed open source

work process worked so well in the that project. He analysed the open source work process used in Linux, and started working on his own project, Fetchmail, in a similar manner to test it in practice. The project turned out to be a success. Raymond wrote an essay called "The Cathedral and the Bazaar", which described his work on Fetchmail and the open source -based work process used in Linux. Seven months after the essay was published, Netscape Communications, Inc., influenced by the essay, decided to open source their Netscape Communicator (later known as Mozilla) internet browser. (Raymond, 1999.) Netscape's browser was losing their market-share to other browsers, mostly to Microsoft's Internet Explorer. The decision to open source the development of their browser under a parallel open development plan was made to give them competitive advantage. (Feller & Fitzgerald, 2000.) Eric Hahn, the Executive Vice President and the CTO of Netscape, contacted Raymond, and asked him to help them design the open source release strategy and licence for the browser. The open source strategy proved to be successful, and Netscape achieved of their goal of preventing Microsoft getting a monopoly status on the internet browser market. (Raymond, 1999.) Raymond co-founded OSI with Bruce Perenis in 1998. OSI realised that all the attention around the decision to open source the Netscape Communicator provided an opportunity to educate people about open source based development process. They held a meeting, where the attendees agreed that the practical and business-based approach Netscape used when deciding to open source the development of Netscape Communicator demonstrated a good way to engage with software developers and users, encouraging them to participate in improving the software. (Open Source Initiative, 2018.) As the previous common term free software from was often understood in a way that individuals or organisations could not gain revenue with it (Fitzgerald, 2006), they wanted to separate this new approach from the Free Software movement by giving it a new label called Open Source, coined by Christine Peterson. The new term was supported and adopted quickly by prominent members of the open source community, such as Linus Torvalds and the founders of Apache, Python and Perl. (Open Source Initiative, 2018.)

The success of open source model has been attributed to the use of licensing. Several different open source licenses have emerged in addition to the GPL, each providing unique distribution terms. Some examples of these are the Lesser GPL, a lighter version of the original GPL, intended for use in software libraries, and the Berkeley System Distribution (BSD), a very lightly restricted license which main requirements are acknowledging and keeping the previous contributors' work. In some cases, the existing licenses were not flexible enough for commercially-oriented OSS projects. Netscape created Mozilla Public License (MPL), when they converted their commercial software to open source in order to avoid existing licenses' restrictions, like forcing each licensor whose code was merged into the Netscape browser to have the same license in the case of GPL. MLP has since been influential, and other corporate style licenses have been created based on it. As OSS started gaining more popularity and corporate involvement started getting more common, more licenses for different purposes have been created. Since then, Free Software Foundation and Open Source Initiative have approved over a hundred diverse licenses. Various non-approved licenses for corporate purposes have been also created. (Fitzgerald, 2006.) According to Open Source Initiative (n.d.), the most popular licenses approved by them are currently Apache License 2.0, 3 and 2-clause BSD licenses, GPL, GNU Lesser General Public License (LGPL), MIT license (MIT), Mozilla Public License 2.0 (MPL-2.0), Common Development and Distribution License 1.0 (CDDL-1.0), and Eclipse Public License (EPL-2.0).

Nowadays, there are over twenty source code repositories and other development and distribution resources for developing OSS (Rajanen & Iivari, 2019). GitHub, one of the major OSS repository platforms, produced a status report based on the data they collected

from October 2019 to September 2020. They reported having over 56 million developers, over 60 million new code repositories, 1,9 billion added contributions, and 72% of the Fortune 50 companies using the GitHub Enterprise service. (GitHub, 2020.) Another popular platform, SourceForge, reported having over half a million OSS projects and over 2,1 million registered users, nearly 30 million visitors, and providing over 2,6 million software downloads in a day (SourceForge, n.d.-d). It has been estimated that even by the year 2008, the total value of the OSS products and services was about 6% of the total value of all software and services, and the adoption of OSS by consumers had saved them about 60 billion dollars in total (Standish Group, 2008, as cited in Rajanen & Iivari, 2019).

2.1.2 OSS development process

Eric Raymond's metaphor of the bazaar is one of the most well-known descriptions of OSS development process (Crowston et al., 2004). He describes two distinct approaches to software development, calling them as the cathedral and the bazaar. By the cathedral, he means a development model where the software is crafted methodically by a group of developers, focusing on stable releases instead of frequent beta releases in order to mitigate the risk of frustrating the users with a buggy experience. Much like how commercial software is usually developed. In the bazaar model, software is developed by a scattered group of developers with different agendas, much like merchants in a bazaar. The bazaar model describes the way how OSS is usually built. Raymond uses the distributed Internet-based development of Linux kernel as the basis for this kind of development model and he credits Linus Torvalds, its lead developer, as the inventor of the model. In this model, beta versions of the software are released very often to expose possible underlying bugs and issues to the users to help with debugging by maximising the person-hours put into it. They assume that if the tester and developer base is large enough, all problems are found quickly and usually someone knows how to fix them, or in other words, "Given enough eyeballs, all bugs are shallow", as Raymond puts it in his Linus' Law. (Raymond, 1999.)

Raymond (1999) also argues that Brooks' Law does not apply to distributed Internet-based software development such as the bazaar model due to its more open community-centric coding style and the access to world-class talent pool of developers. Brooks' Law means that adding more developers to a late software project only delays it further. According to Brooks (1995, as cited in Raymond, 1999), the project's complexity and communication costs increase with the square of the number of developers while the actual work that is done only increases linearly. Raymond (1999) argues that the success of Linux proved that massive distributed software projects can work effectively.

Bezroukov (1999) criticises the bazaar model for making questionable claims about OSS process which have become generally accepted. One example of such is the statement that the Brooks' Law does not apply to Internet-based distributed software development. He argues that the non-applicability requires a fully functional prototype of the program or that all the architectural issues of the software have been already solved. He also argues that the claim that the bazaar model was something completely new was false. He sees it as a logical evolution of its free software origins started by the Free Software Foundation's GNU project. He suggests that the OSS process can be explained better by describing it as a form of academic community, where the participants seek for recognition and status by contributing to the community. He argues that this could have been due to both Free Software Foundation's and Torvald's strong connections to the academia. He also critiques the bazaar for representing status competition as a solely positive thing, and suggests it can lead to negative things, such as unfair hierarchies

caused by favouritism, increasing bias in code peer reviewing, and the dangers of burnout especially when trying to compete with rival software.

Fitzgerald (2006) argues that even though the bazaar metaphor has been a common way to describe the open source software process, OSS development has since transformed as corporations have recognised its commercial potential. He describes this new process as less like a bazaar and more organised, as the OSS spreads to vertical product domains which require more strategic planning because the requirements of the software are not understood universally by the developers. He argues that as the planning phase of OSS was characterised as the developers "scratching one's itch" in the bazaar metaphor, organisations have since joined the game and started using strategies to gain competitive advantage utilising open sourcing, by for example implementing software by using same kind of fast beta release iterations and community support but with more organised project management, or by paying developers to work on OSS.

2.1.3 Motivations for participation in OSS development

The joy of programming and personal skill development have emerged as important motivations of OSS developers in some empirical studies (Andreasen et al., 2006; Hars and Ou, 2002). Hars and Ou (2002) investigated the motivations of people who participate in open source software development by conducting a survey with 79 respondents from the open source community. The authors identified two types of participation motivations; internal and external. Internal motivation was divided into self-determination (the feelings of fulfilment and competence one gets from coding), altruism and community identification. External motivation was divided into future rewards (such as selling products, developing skills, self-marketing and peer recognition) and personal need. According to the authors' analysis of the survey data, different kinds of groups of developers with different motivations participate in OSS development. Groups like students and hobbyists were the most internally motivated groups, while contracted and salaried programmers were the most externally motivated groups. In total, external motivations were found to have a greater weight than internal motivations, even though internal motivations had an impact too. Among all respondents, the most common motivations were developing skills (88.3% of the respondents were motivated by it) and self-determination (79.7%). (Hars & Ou, 2002.) Andreasen et al. (2006) achieved similar results in their survey and interview-based study about OSS developers' attitude towards usability, where they also examined the motivations of the participants. In their study, 21 of the 24 respondents claimed contributing being intellectually stimulating and strengthening free software as their motivations for participating. 18 of the participants found improving skills as a motivational factor.

Hertel et al. (2003) examined the motivations of Linux kernel's contributors through the lens of social sciences by conducting a web-based survey with 141 respondents. The authors used two motivational models from social science literature to investigate the motivations of the developers in a systematic way. The used models were designed to explore the incentives to participate in social movements and the motivational processes of geographically distributed work groups. The survey's questions were derived from the used models and from discussions within the Linux community. Based on the survey's results, the authors argued that the motivational processes of developers of OSS projects resemble the motivational processes of members of other social communities such as social movements, and the motivations of OSS developers can be explained by using existing psychological theories. The authors identified seven main motivational factors in the Linux community and used them as a basis for the survey:

- General identification as a Linux user
- Specific identification as a Linux developer
- Pragmatic motives like improving used software and career advancement
- Motives related to reactions of people such as family members, colleagues, or friends
- Social and political motives such as supporting free software and socialising
- Hedonism
- Motivational obstacles related to losing time to activities related to Linux

All of the factors were found to get high mean scores and they correlated positively with the willingness to participate activities related to Linux, but some of them were found to be more predictive of engagement than others. The engagement was found to be especially determined by their identification as a developer of Linux, pragmatic motives, and by their tolerance of investing time. The authors also examined motivational aspects of distributed team work in the community. Some parts of the software development was found out to have been performed by spontaneous teams. The most important distributed team work -related motivational factors were the participants' perceived indispensability of their contributions to the project, a high feeling of self-efficacy, and a high evaluation of the team's goals. (Hertel et al., 2003.)

2.1.4 The hierarchical structure and decision-making in OSS projects

OSS projects are usually governed by one or a small group of active core developers. Having a 'benevolent dictator' who has the final say in decisions, often supported by a group of high ranking of co-developers with their own areas of responsibilities, (for example subsystems assigned to them) is a common governance model in OSS. The leader usually consults the co-developers when making decisions, especially if they concern areas of code that are assigned to co-developers. Some projects favour a more democratic governance model, where the project is governed by a core developer voting committee. (Raymond, 1998.) In some projects, decision-making has been extended to the whole community by consensus-based governance (Gacek & Arief, 2004). The so-called 'rotating dictatorship' is another style of governance. In it, the leadership of the project is passed occasionally from a core developer to another. (Raymond, 1998.)

Raymond (1998) describes ownership of an OSS project as having the right that is acknowledged by its community to re-distribute the modified versions of the software. He argues that there are three ways to acquire ownership of an OSS project: by founding it, having the ownership gifted to you by the previous owner, or finding a project which previous owner has abandoned and asking the permission to assume control of the project, or if the previous owner is not found, announcing to the community that you are going to take control of it for it if there are no objections.

Open source culture has been characterised as a gift culture, where the social status of individuals is determined by the gifts they give away to the community. Members of an OSS community can gain social capital and power by giving contributions to the project that the community deems valuable. (Bergquist & Ljungberg, 2001; Raymond 1998.) The gift economy interpretation of open source communities has been compared to the way how academic community works, where one trades knowledge for reputation by contributing to the existing research (Raymond 1998). Peer reviewing of code contributions in OSS projects have been also compared to how peer reviewing is done in the academic community (Raymond 1998), and how it can be understood as a way of

organising power relationships within the members of a community (Bergquist & Ljungberg, 2001).

OSS projects have been described to typically have a meritocratic "onion-like" hierarchical structure consisting of several layers which represent different types of community members (Aberdour, 2007; Crowston et al., 2004; Gacek & Arief, 2004).

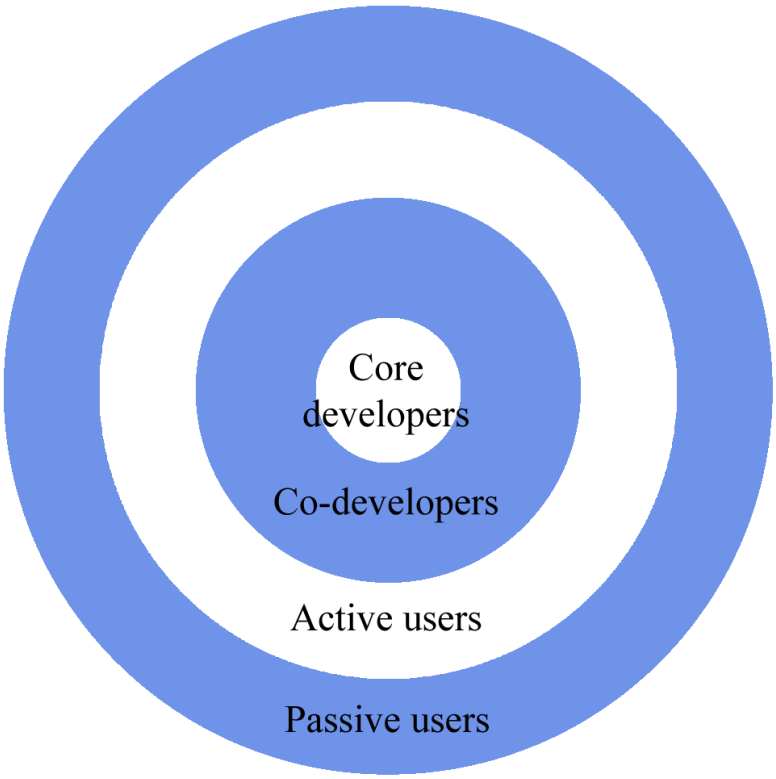

Figure 1. A representation of an onion-like power hierarchy of an OSS project. Adapted from Crowston et al., 2004.

The onion model (Figure 1) consists of core developers, co-developers, active users and passive users. The number of involved people increases the farther we go from the core. The core developers are typically a small group that manages the project and contributes most of the code. The core developer group usually stays small, because a high level of communication is needed, which would be difficult if the group would be too big. Co-developers are a bigger group of developers who casually contribute and review code contributions, such as bug fixes. (Crowston et al., 2004.) They add features to the code and maintain them, often picking features that are defined in the project's road map or they want to implement (Aberdour, 2007). The users are divided into two of the biggest groups; active and passive users. Active users are a group of users who do not contribute code, but they help by testing the newest releases and submitting bug reports and feature requests. Passive users are a group of users who do not actually contribute to the project in any other way than using the software. The roles of community members can change from time to time. (Crowston et al., 2004.) Advancing towards the core developer group from the outside usually happens in a meritocratic way. Developers can increase their perceived merit by contributing to the project and gain decision-making power along the way, though this transformational process varies from project to project depending on timing, the project's structure, and other possible obstacles to overcome. Participants'

roles can also change to the other way, like for example core developers can change their role to co-developers. (Gacek & Arief, 2004.)

The process of newcomers gaining decision-making power by giving gifts has been observed empirically (Ducheneaut, 2005; von Krogh et al., 2003). Von Krogh et al. (2003) conducted a clinical study of an OSS peer-2-peer software project Freenet. The authors described the community to be governed by core developers who have the commit rights. They examined how can newcomers become a part of the community and eventually gain power within the community. They suggested several actions that can help in getting accepted by the community based on the actions made by the successful joiners. Examples of such are observing the community discussions and learning about the project before making the first contributions, starting out by participating in the existing discussions instead of proposing own technical solutions, and presenting "feature gifts", often related to their own area of expertise. They suggested that specializing can be beneficial to the developer, because it allows them to focus on their area of expertise, making contributing easier. New feature gifts can also act as starting points for contributions to other newcomers. (Von Krogh et al., 2003.) Excessive amounts of features have been criticised for contributing to usability problems, though (Nichols & Twidale, 2003).

Ducheneaut (2005) acquired similar results when he examined the socialisation process of new developers in a case study of the open source programming language Python's community. He analysed the project's code commit history and mailing list message data in order to figure out how newcomers gained decision-making power in the community. A total of 284 participants were examined, and 136 of them had only posted only one message to the community and left. The participants who left were not included in the analysis because they did not become a part of the community. He compared the influence gaining trajectories of individual newcomer developers by examining their commit and message histories, especially focusing on how the successful developers gained power in the community. He explained that many of the newcomers who successfully advanced to developers often took similar steps when they managed to gain power in the community:

1. Observing the development before contributing
2. Reporting bugs and suggesting new patches
3. Fixing bugs
4. Managing a module-sized part of the software
5. Developing the module and gaining support for it in the community
6. Getting the approval from the core developers to add the module into the software

Ducheneaut (2005) argued that newcomers can assimilate into the community's culture by "lurking" for a period of time before starting to contribute to the project, but eventually they will have to start building an identity as a "software craftsman" for themselves in order to be noticed by the core developers. He suggested that this can be achieved by demonstrating one's merit by participating in technical discussions and eventually submitting bug reports and proposed fixes. Ducheneaut (2005) argued that gaining influence in the community is as much of a political process as it is about technical expertise: one must understand how the politics of the community work and gather allies in order to succeed.

2.2 Usability

The International Organization for Standardization's ISO/IEC 25010 and ISO 9241 standards define usability as the extent to which a system can be used by specific users to achieve their specified goals with efficiency, effectiveness, and satisfaction in a defined use context (International Organization for Standardization, 2011; International Organization for Standardization, 2018). In the ISO/IEC 25010 standard, usability is one of the eight quality characteristics of the product quality model, and it is divided into sub-characteristics appropriateness recognisability, learnability, operability, user error protection, user interface aesthetics and accessibility (International Organization for Standardization, 2011). As an area of research, usability can be considered a part of human-computer interaction (HCI) research (Väänen-Vainio-Mattila, 2011, p. 123).

Usability expert Jakob Nielsen defines usability by dividing it into five different sub-attributes:

- Learnability
- Efficiency
- Memorability
- Errors
- Satisfaction

Learnability means how easy it is to learn to use the system. Efficiency refers to how productively advanced users can operate the system. Memorability means how easy it is to remember to use the system effectively after a pause if the user has already learned to use it before. Errors refers to how often users make errors while using the system, how critical they are, and is it possible to recover from them with minimum effort. Satisfaction means how enjoyable it is for the users to use the system. (Nielsen, 1994a, pp. 26-33.) Nielsen (1994a, pp. 24-25) argued that usability by itself is a part of a larger concern of system acceptability, which represents the question if the computer system is good enough to satisfy the needs of users and other stakeholders. The overall acceptability consists of social and practical acceptability. Social acceptability determines do the users find the way the system is works socially acceptable, for example ethics-wise. Practical acceptability consists of sub-factors like cost, reliability, support, compatibility with other systems, and usefulness. Usefulness refers to the question can the system achieve a desired goal of the user, and it is divided into utility and usability. Utility determines if the system can functionally perform the needed tasks to achieve the goal, and usability refers to how well the users can use the system as measured by the attributes of learnability, efficiency, memorability, the lack of errors, and satisfaction.

Modern usability design and research are based on the science of ergonomics. Ergonomics advanced greatly during the World War II, when the warring nations needed to develop their war machines competitively. In the 1980's, a new branch of science, called human-computer interaction, emerged, and usability has been acknowledged as a significant component in development of products since then. The International Organization for Standardization created the first international standards for usability (ISO 9241 and ISO 13407) in the 1990's. (Väänen-Vainio-Mattila, 2011, pp. 102-103.) Also around that time, using usability inspection as a way of improving usability of computer systems started getting more popular (Nielsen, 1994b). Modern usability research can be described as an interdisciplinary science that combines elements from information processing science and psychology. It also utilises methods from other

disciplines, such as marketing, linguistics, and sociology. (Väänen-Vainio-Mattila, 2011, p. 103.)

Väänen-Vainio-Mattila (2011, p. 104) argues that usability of systems is important because of humane and economic reasons, because used systems can improve the quality of life of people and give them joy. Bad usability has been suggested to increase the costs of product design and support as well as having a negative effect on the competitiveness of the software (Bias & Mayhew, 2005, as cited in Väänen-Vainio-Mattila, 2011, p. 104). Usability can be improved and designed by using user-centered design and usability engineering methods (Rajanen et al., 2012).

2.2.1 User/human-centered design

HCI supports the philosophy of user-centered design (UCD) as a way of developing software usability. It is a way of developing software by involving its end-users throughout its life cycle in order to make software that will fulfil their requirements. (Viorres et al., 2007.) The aim of UCD is to gather user knowledge that is more truthful than just pure intuition of the designers to the design process. The analysis of use context, eliciting the users' requirements and needs, and iterative evaluation of produced design concepts with the users are some of the core areas of work in UCD. (Väänen-Vainio-Mattila, 2011, pp. 102-105.)

ISO 9241 defines human-centered design as an "approach to systems design and development that aims to make interactive systems more usable by focusing on the use of the system and applying human factors/ergonomics and usability knowledge and techniques" (International Organization for Standardization, 2019). The standard provides recommendations and requirements for implementing human-centered design activities to interactive systems throughout their life cycle. The terms user-centered design and human-centered design are often used interchangeably in practice. The term human-centered design was preferred to UCD to in the standard to emphasise that the standard addresses also concerns of stakeholders of the system that are not usually considered as users. (International Organization for Standardization, 2019.) The ISO/TR 16982 standard has listed many kinds of UCD methods. Examples of these are watching users in their work environment, user surveys, user interviews, thinking aloud user testing, collaborative design and evaluation with users and developers, and expert evaluations. (International Organization for Standardization, 2002, as cited in Väänen-Vainio-Mattila, 2011, p. 111.)

It has been argued that UCD is a vague concept (Hedberg et al., 2007), and there is confusion regarding its practices, principles and goals (Iivari & Iivari, 2006). Iivari and Iivari (2006) examined user centeredness of UCD as a concept composed of four dimensions: user focus, work-centeredness, user participation, and system personalisation. User focus refers to focusing on typical or individual users of the system. Work-centeredness means focusing on the work done by the users and examining the users as workers instead of individuals. User participation refers to the extent the actual users of the systems are involved in the design process. System personalisation refers to the extent that system can be modified to accommodate individual users' preferences. (Iivari & Iivari, 2006.) According to Hedberg et al. (2007), UCD methodologies emphasise different dimensions of user centeredness. They argue that although there are many different kinds of UCD methodologies that focus on different areas, they all share similar emphasis on the importance of understanding the users, their tasks, and the context of use (Hedberg et al., 2007). Having users involved in the design process has been

criticised for being risky in a way that listening too much what they want may lead to systems with many features but without proper support for the user tasks (Norman, 2005, as cited in Väänen-Vainio-Mattila, 2011, p. 109).

2.2.2 Usability engineering methods

According to Nielsen (1994b), empirical usability engineering methods, such as user testing, are usually the most common way of evaluating user interfaces, but their disadvantage is that it can be expensive to hire the sufficient amount of testers for every iteration of an evolving design of a system. He suggests usability inspection methods such as heuristic evaluation and cognitive walkthrough as a cost-effective way to conduct usability engineering. He also recommends combining user testing with different usability inspection methods to complement each other, since all of the methods can overlook different kinds of usability problems.

Cognitive walkthrough

Cognitive walkthrough is a usability inspection method, where usability evaluators go through the user interface and try to simulate a user's problem solving process at every phase of the of the system's dialogue, and assess if the user's current objectives and memory content lead towards the next defined correct action (Nielsen, 1994b). Cognitive walkthrough is focused on the user's cognitive issues like learnability (Holzinger, 2005). Holzinger (2005) describes helping designers to understand the user's perspective, the effective detection of issues related to dialogue with the system, and not requiring a functioning prototype of the system or test users, as its advantages. As its disadvantages, he mentions emphasis on low-level details, not having the end users involved, and possible dullness and inherent bias as a result of selecting the tasks improperly.

Heuristic evaluation

Heuristic evaluation is a usability evaluation method where inspectors look at the user interface and assess what is good and bad in it, ideally by evaluating it against a certain set of rules or guidelines. The inspectors go through the graphic user interface (GUI) multiple times in a single session, evaluating the interactive elements against the selected guidelines. In the end, the inspectors produce a list of usability problems linked to the specific usability principles they violated. (Nielsen, 1994a, p. 155-159.) According to Holzinger (2005), the used set of heuristics should be selected carefully to fit the specific system that is being designed. He mentions the use of recognised and accepted principles, having usability considerations early and throughout the development, being intuitive to use, and effective detection of both minor and major issues, as the advantages of this usability inspection method. He considers detachment from end users, unreliable detection of issues that are specific to certain domains, inability to design for unknown user groups, and possibly not evaluating the complete design of the system, as the disadvantages of heuristic evaluation. (Holzinger, 2005.) Heuristic evaluation has also been critiqued by questioning if focusing on minor usability problems reduces the effectiveness of the technique (Sears, 1997).

Jakob Nielsen and Rolf Molich (Molich & Nielsen, 1990, as cited in Nielsen, 1994a, pp. 19-20) propose a set of ten general usability principles that can be used for systematic usability assessment of a GUI design by using heuristic evaluation:

1. Simple and natural dialogue
2. Speaks the users' language
3. Miminise the users' memory load
4. Consistency
5. Feedback
6. Clearly marked exits
7. Shortcuts
8. Good error messages
9. Prevent errors
10. Help and documentation

Simple and natural dialogue means that irrelevant or rarely needed information should be filtered out in order to highlight relevant information, and all the visual information should be presented in a logical order. Speak the users' language refers to using clear language that is apt for the intended user group instead of computer jargon. Minimising the users' memory load means aiming to design the system in a way that the user should not have to memorise information between different parts of dialogue, and allowing the user to access the user manual whenever needed. Consistency refers to using consistent terminology and eliminating possible guessing whether different words or actions mean the same thing. Feedback means keeping the users constantly updated about what the system is currently doing. Clearly marked exits refers to implementing clearly understandable exit actions to allow leaving an unwanted state. Shortcuts refer to having quick actions that can save time for expert users which do not distract new users. Good error messages means having error messages that are written in a way that the users understand them and suggest possible solutions for the problem. Preventing errors means designing the system carefully and preventing errors from occurring. Help and documentation refers to having a documentation that can be used for searching for help for possible problems that the users can encounter. (Molich & Nielsen, 1990, as cited in Nielsen, 1994a, pp. 19-20.)

According to Nielsen, the main objective of heuristic evaluation is to detect the usability issues in a GUI design so they can be addressed as a part of iterative design process (Nielsen, 1994a, p. 155). He explains that these heuristics can be used to detect a large percentage of usability problems in user interfaces. Using them fully accurately requires some expertise, but even novices can also find many problems by using heuristic evaluation. Nielsen recommends especially novice evaluators to use other usability inspection methods such as thinking aloud test to supplement the results of heuristic evaluation. (Nielsen, 1994a, pp. 19-20.) It is also recommended to have multiple different people to conduct heuristic evaluation alone to make it less likely to miss possible usability problems (Nielsen, 1994a, pp. 19-20) and to ensure unbiased evaluations (Holzinger, 2005).

User testing

Nielsen (1994a, pp. 165-200) considers testing with real users the most essential usability method, perhaps even indispensable, because it allows to get direct feedback about how people use the system and what kind of problems they encounter when using it. User tests are often conducted in a special usability laboratory setting, but they can be also conducted in normal office settings that are converted into temporary usability laboratories. Special equipment is not mandatory, as user tests can be conducted even only with a notepad. A typical user test consists of four phases: preparation, introduction, the test, and debriefing. (Nielsen, 1994a, pp. 165-200.)

Nielsen (1994a, pp. 165-200) divides user testing into two categories: formative evaluation and summative evaluation. Formative evaluation is used to improve the design of a user interface in an iterative manner by evaluating what is good and bad in it. Summative evaluation, on the other hand, is focused on evaluating the user interface's overall quality. It is used for example when trying to decide between alternative user interfaces, or as a way of conducting competitive analysis by evaluating competitive user interfaces. Thinking aloud testing is a typical formative evaluation method. In thinking aloud testing, users use the system and verbalise their thought process, giving the evaluators insights on possible usability issues. There are also similar other versions of user testing, such as constructive interaction (two test users use the system together and talk about it), retrospective testing (recording the test session and after it the test user discusses the video), and the coaching method (a coach gives advice to the user during testing). (Nielsen, 1994a, pp. 165-200.)

Although Nielsen (1994a, pp. 165-200) argues user testing to be very important, he warns that special attention must be paid when testing to achieve replicable results and focus on the right usability issues that are encountered in the real environment instead of only in a testing laboratory. As the variance of individual test users' results can be great, he recommends getting a sufficient amount of testers and performing statistical analysis to achieve more reliable results. He argues that also validity problems can occur when types of test users or test tasks that are not representative of the actual users and tasks are selected, or time constraints and social influences are excluded from testing.

2.3 Usability in OSS projects

Even though there is lots of open source software available and the use of OSS has been growing, OSS tends to be used the most by technically advanced users, while average users usually prefer proprietary software. One of the main reasons proposed for this is the perception that OSS has weaker usability. (Nichols & Twidale, 2003.) It has been often suggested that usability is a problem in open source software (Andreasen et al., 2006; Feller & Fitzgerald, 2000; Lisowska Masson et al., 2017; Nichols & Twidale, 2003). OSS projects often emerge from personal need of the developers (Moody, 2001, as cited in Crowston et al., 2004; Raymond, 1999; Vixie, 1999, as cited in Crowston et al., 2004), and the developers are also users of the software (Andreasen et al., 2006; Crowston et al., 2004; Nichols & Twidale, 2003). As the user base of OSS grows and an increasing amount of non-expert users start using it, there is a need for additional usability considerations (Feller & Fitzgerald, 2000; Nichols & Twidale, 2003; Rajanen & Iivari, 2019).

2.3.1 Usability issues in OSS

Nichols and Twidale (2003) suggest several possible reasons why usability issues are prominent in OSS:

- Developers do not represent typical users
- Usability experts do not involved in OSS projects often
- OSS projects do not have the resources for high quality usability work
- The working incentives in OSS are more suited for improving functionality rather than usability
- OSS is more prone to feature bloat than commercial software
- OSS's tendency to focus on power instead of simplicity

- Usability issues are more difficult to specify compared to problems related to functionality
- It is more difficult to conduct usability design after the coding has already begun
- Commercial software determines what is cutting edge GUI design due to its popularity and the OSS can only attempt to follow it

Software developers often fail to see what kind of usability problems other kinds of user than themselves may encounter, and the developers of OSS are not always the typical users. In commercial software development, usability experts are often hired to bridge the gap between the developers and the users. In OSS development, the OSS communities often lack the resources to hire usability experts to conduct high quality usability work, and it is somewhat rare for usability experts to get involved in OSS projects. (Nichols & Twidale, 2003.) While the usability experts usually have the authority to represent the needs of users in commercial software projects (Nichols & Twidale, 2003), they are often struggling to get their voice heard in OSS projects (Çetin et al., 2007). OSS developers are often driven towards developing features and functionality instead of focusing on usability and simplicity. This can be explained by the voluntary nature of OSS development. As developers contribute to OSS projects by selecting the topics and parts of software that interest them, many of them may not be interested developing modifications related to usability. This and the fact that there is always an incentive to add more code from other developers and never to delete parts of the it can lead to feature bloat and increasing complexity of the software, which can be detrimental to usability, especially for novice users. Usability issues can be difficult to specify and evaluate compared to functional problems, and usability work can also be hard to divide to several different developers. Usability design work tends to most effective when it's implemented as early as possible, but in OSS, the developers tend to rush straight to coding. Usability innovations are usually made by commercial software, and the OSS has often focused on following user interface design of brand leaders instead of developing them even further. (Nichols & Twidale, 2003.)

Andreasen et al. (2006) examined OSS developers' opinions about usability and how usability work is performed in OSS projects by conducting a survey and interviews of OSS developers and usability evaluators with 24 participants from OSS projects without corporate involvement. The respondents generally considered usability as important, as 83% of them regarded the importance of usability high (from high to very high and extremely high), and only 13% of them considered it moderate and 4% considered it to have a slight importance. One of the interviewed developers who had ranked usability to have extremely high importance argued that some of the OSS developers see usability related work as boring tasks compared to coding features. The developers were also asked about what they thought of usability experts contributing in OSS. The authors argued that many of the developers were reluctant to involve usability experts in the development and saw them as a threat to their perception of the democratic nature of OSS, if the usability experts are the only ones with expertise about the subject matter. One interviewee explained that usability experts telling the developers what to program would be another issue, since they do not like to be told what to code. The interviewed developers were more accepting towards having external usability experts conduct usability evaluations to the software instead of participating actively in its design process. The authors also asked the participants about what stage of the development process usability activities should be performed and received mixed answers. Only 5 of the 23 respondents (one of the initial 24 of them was not sure about what to answer) saw usability as an iterative process continuing throughout the project. Applying usability work in the beginning of the project was the most common answer (12 answers), and during the testing phase was answered

5 times. Common sense was found to be the primary usability evaluation method among the participants, as 19 of 24 them answered that they followed common usability conventions and usability guidelines. The use of external professional usability work was not as common (10 of 24 participants); some of the projects had used usability inspections, but they were rarely conducted by professionals.

The tools used in OSS development have been also critiqued for being difficult to work with when trying to improve usability (Çetin et al., 2007; Nichols & Twidale, 2006). Nichols & Twidale (2006) examined the bug reporting tools of Mozilla and GNOME projects and found problems related to reporting usability issues. For example, they argued that it can be difficult or cumbersome to express encountered usability problems or propose GUI solutions textually (instead of using discussing them face-to-face or using GUI mock-ups), and some usability issues are related to a sequence of actions instead of a single interactive element, making it difficult to describe it. They also argued that discussing about usability issues can be complex and contentious due to the fact that usability issues can be perceived as subjective. Another issue according to them is managing the complexity of fixing usability bugs. For example, fixing a single usability issue can have undesired effects on the overall interface design. Çetin et al. (2007) also conducted empirical research on usability experts' involvement in OSS and critiqued the usability issue reporting tools for not supporting multimedia format usability reports.

Several ways to improve the usability of OSS have been suggested (Çetin et al., 2007; Nichols & Twidale, 2003; Nichols & Twidale, 2006; Zhao & Deek, 2005). Nichols & Twidale (2003) suggested that involving software companies or the academia could be beneficial for usability. Software companies could be used for developing the GUI for an OSS application with their usability resources. In academy involvement, HCI students could participate in actual OSS projects, offering their expertise for example in the form of prototypes. The authors argued that the students could get practical training this way, and the OSS platforms might have to eventually evolve in a way that is more supportive towards usability work. They also mentioned that usability education and evangelism may be needed to convince typical OSS communities about its benefits in order to become more accepting towards it. Çetin et al. (2007) suggested that making the developers more aware about basic usability principles would be beneficial, because the developers could then evaluate the usability of the software themselves, and this would also increase the mutual understanding between the developers and the usability experts. Getting typical users more involved in usability issue reporting has been suggested as a one way of improving usability (Nichols & Twidale, 2006; Zhao & Deek, 2005). Nichols and Twidale (2006) argued that creating an infrastructure that would support usability issue discussion would potentially allow the developers to engage with a larger group of users, including the passive users who do not normally report bugs. They also suggested that it could be beneficial to have the developers apply condensed knowledge from HCI, such as Human Interaction Guidelines and concepts like Fitt's Law, but argued that a more effective solution would be creating a community that would encourage HCI experts to participate in development. According to Bach et al. (2009), there are three ways of how usability experts get involved in OSS projects: they are either paid to work in them, volunteers who find a suitable project, or they are paired with an OSS project by using a usability work matchmaking service like OpenUsability.org.

2.3.2 Usability practitioners' barriers to contributing in OSS projects

The core ideas of user-centered design emphasise that it is difficult for software developers to design for typical users (Nichols & Twidale, 2003). Many kinds of user-

centered design methods exist in the HCI field, but they have not been used widely in OSS development (Bødker et al., 2007). Nichols and Twidale (2003) propose involving user-centered design practitioners to connect the developers to the average users by using techniques like participatory design and usability engineering. It can be challenging, though, because OSS has been described having characteristics that do not mix well with HCI philosophy, such as geographical distribution, code-centricity, lack of usability expertise and resources, and a culture that often feels unfamiliar to usability practitioners (Nichols & Twidale, 2006). Muehling and Reitmayr (2006, as cited in Çetin et al., 2007) identified some challenges related to shifting towards a more user-centered approach in OSS development. They argue that in OSS, the intended target audiences and their requirements and tasks are often not defined clearly. They also suggest that it may be difficult for usability experts to get their voice heard, because they may need to convince more people about the usability work than in traditional development. (Muehling & Reitmayr, 2006, as cited in Çetin et al., 2007.) Çetin et al. (2007) argue that it can be problematic that usability experts' active participation in OSS projects slows down the software's time-to-market.

Rajanen and Iivari (2019) argue that there are OSS projects in need of usability work and usability experts who are willing to contribute to them, but their effective collaboration requires that the OSS projects in need of usability work decide to incorporate the usability activities into their road map, and the usability experts need to be able to find those kinds of projects and manage to convince the decision-makers of the importance of usability. According to Nichols & Twidale (2003), usability practitioners often lack the technical skills of traditional OSS developers, and it can lead to them not taken seriously by the community. Andreasen et al. (2006) interviewed five usability professionals involved in OSS development. The usability professionals argued that almost all issues they encountered when working with OSS developers stemmed from lack of trust, which often caused the developers to reject their suggestions. Andreasen et al. (2006) argued that the distributed nature of OSS where the developers rarely meet in person makes people judge each other based on their past merits. It can be difficult for usability practitioners to demonstrate their value and build trust, because usability work is more difficult to measure than functional contributions. Andreasen et al. (2006) and Trudelle (2002, as cited in Andreasen et al., 2006) argued that usability practitioners must be willing to build trust through merits to gain decision-making power in the meritocratic culture of OSS. The interviewed usability professionals suggested that external usability experts could also establish trust by meeting the developers face-to-face, for example in an OSS conference. They stated that their work environment improved after trust was established. (Andreasen et al., 2006.) It has been also suggested that usability improvement proposals can be rejected for the reason that the project lacks resources to implement them, as Zhao and Deek (2005) encountered this issue in a case study of a usability intervention on an OSS project. Çetin et al. (2006, as cited in Çetin et al., 2007; Çetin et al., 2007) examined the impact of usability activities done by usability experts in OSS projects, and argued that the timing of the usability intervention affects their influence on the project. According to them, the earlier the usability experts join the project, the higher their chances of getting accepted in the community and having an influence on the user interface are.

OSS environments that would be more suitable for usability work have been theorised on a conceptual level (Bach et al., 2009). Bach et al. (2009) addressed issues of usability experts gaining trust and merit and the OSS platforms' inadequate support for usability work in their study, where they produced GUI mock-ups for modifying CodePlex OSS community platform to be more suitable for usability work. Their aim was to create concepts of an OSS platform that would make usability experts feel welcome. They used

previous literature and interviews of usability practitioners as the basis for their proposed changes. For promoting building trust and merit, they proposed giving different work spaces to developers and usability practitioners to make it clear that usability work is valued, and creating a space where the usability practitioners can share usability concepts and discuss about them with others, and demonstrate their expertise by describing the rationales that the designs were based on. They also proposed galleries that would display the top GUI designs on the front page. In order to bring best usability practices to the platform, they proposed adding direct support for usability tasks, such as persona descriptions, scenarios, design iterations and user stories in the usability work space.

High quality OSS requires a sustainable community (Aberdour, 2007). Many volunteer-based open source software projects are dependent on newcomers joining the project continuously (Steinmacher et al., 2015). OSS communities need to engage, motivate, and retain new developers in order to cultivate a sustainable community of software developers (Qureshi & Fang, 2011, as cited in Steinmacher et al., 2015). It can be difficult for newcomers to contribute to ongoing OSS projects because they often encounter different kinds of obstacles when trying to join the community (Dagenais et al., 2010; Steinmacher et al., 2015). Dagenais et al. (2010) identified different kinds of obstacles newcomers of OSS projects can encounter by interviewing 18 newcomers of 18 different projects. Obstacles, such as poor quality feedback (or the lack of feedback) from the community or inadequate project documentation were found to make joining and integrating into an OSS community difficult for newcomers. Steinmacher et al. (2015) conducted a systematic literature review of obstacles newcomers encounter when entering OSS projects. They found 20 empirical studies that provided evidence on different kinds of obstacles, and classified the obstacles into five categories: social interactions, finding a way to start, the newcomers' knowledge, technical issues, and documentation. Obstacles related to social interactions (such as not receiving answer from the developers in time, receiving an improper answer, or the lack of interaction with the developers) were the most common, as they were present in 75% of the studies. Newcomers' previous knowledge was the second most common obstacle category, and the lack of technical expertise was the most common issue related to this obstacle. Steinmacher et al. (2015) suggested that having domain knowledge combined with technical skills and social interaction with the developers may help when joining a new OSS project. According to them, discussions of OSS developers often revolve around artefacts that reflect domain and technical knowledge, and through the outcomes of those discussions, the developers and the newcomers can evaluate if the skill level required for contributing to the project is apt. The authors also suggested that the OSS project communities themselves could work on becoming more receptive towards newcomers and taking newcomers into account by for example keeping the code easily readable and the documentation up-to-date. Viorres et al. (2007) argued that also the open source software tools used for creating OSS, such as compilers or file editors, can be difficult to work with even for developers. They mentioned issues like a modular way of producing OSS can make it more difficult to install, use and maintain the software, the limited or fragmented documentation, and not taking backwards compatibility into account.

A typical distributed and technologically-driven OSS project culture is at odds with corporate processes and usability engineering methods, making it difficult for professional usability practitioners to work in OSS projects (Benson et al., 2004). Benson et al. (2004) studied what kind of difficulties corporate usability experts faced when working on OSS projects. They examined empirically NetBeans, GNOME and OpenOffice projects, which had corporate usability involvement. They identified challenges, such as communication problems between the developers and the usability teams, confusion about the target user groups, and usability teams lacking in decision-

making power in GUI design issues. The communication tools usually supported usability discussions poorly, as the discussions were fragmented to several places and complex bug databases could be intimidating to non-technical contributors. In the case of GNOME, the usability practitioners' role was more like a usability bug reporter than actual designer who would iteratively design user interfaces. User interface modifications were usually already designed by the developers before they asked for assistance from the usability team. Benson et al. (2004) argued that in order to conduct professional usability work in OSS effectively, integrating a fitting usability methodology for OSS processes and defining a centralised and decision-making process would be important.

Bødker et al. (2007) conducted a study, where they tried to bring user-centered design processes to the OSS project community of TYPO3, an enterprise content management system. The study was action research based, in which they aimed to change the community towards becoming more accepting to usability considerations. They performed two usability interventions. In the first one, they decided to gather user knowledge, since they noticed that the developers had no clear idea of what are the typical users of the software. They created a HCI discussion list where they discussed usability issues and recruited usability "ambassadors", members of the developer community who were interested in usability. Although the ambassadors did good work in the discussion list, their lack of knowledge of usability concepts became a problem as discussions often shifted towards plain general observations of usability, so the authors decided to change their approach by conducting another intervention. In the intervention, they shared a set of usability heuristics to the community which were intended to provide a common vocabulary of usability to all developers and to educate them. The authors described the interventions as challenging, because they met resistance towards usability work from the developers. A common counter-argument from the developers towards usability work was that why should they take end-users into account, because they were working only for fun without getting paid. The authors suggested that the original ideology of OSS may be at odds with user-centered development due to its voluntary nature. As for the results of the study, they concluded that they managed to change the community by introducing and promoting usability discussion and work, and changing some of the developers' attitudes towards the end-users to more positive.

Lisowska Masson et al. (2017) conducted a case study where they advocated for consistent use of GUI design principles in a large-scale learning management OSS project called ILIAS. The power structure of the OSS project in question followed the typical onion model consisting of developers and users of different ranks. The core developers and the project manager held bi-weekly meetings where they discussed new proposed features. The authors created a toolset called Kitchen Sink which purpose was to address usability issues by helping usability practitioners to integrate into the developer community and reducing the effort of developers to implement usability improvements. Its main goals were:

- Encouraging developers to value the work of usability practitioners
- Providing a taxonomy of the GUI components of the software and defining clear and effective ways of using them
- Providing a way for usability practitioners to contribute to the project effectively
- Making it possible to conduct automated test for some of the guidelines

A prototype of Kitchen Sink was made and presented to the developers. It was met with mixed reactions. There was interest in the project, but some raised concerns, such as doubts about its funding, the slow pace of designing it, and resistance towards major changes. A veteran developer of the software who had used the software for 15 years and

had done some usability work on it emerged as a supporter of the project on a condition that some changes were made to the project according to her feedback. Her help proved to be valuable, because she was able to determine what kind of changes would cause harmful ripple effects to other parts of the software. Two major changes were made to the initial plans. The initial plan was using a holistic top-down approach for implementing GUI changes, but it was changed to bottom-up approach which consisted of implementing small changes. This was decided because the authors wanted to demonstrate trust to the community and to address the issues of limited resources of the developers and the large userbase affected by changes. The authors created a taxonomy of the existing GUI components of the software which were used for suggesting usability improvements and a template for Kitchen sink entries which can be used when proposing new GUI elements to be presented in the core developers' meetings. The taxonomy consisted of 130 entries and 361 guidelines on how the developers can improve them. Kitchen Sink was eventually accepted by the core developers, and they reserved a time slot for presenting its entries in the bi-weekly meetings. In two months after its acceptance, 11 Kitchen Sink entries had been accepted for implementation and 2 were rejected. The authors suggested that even though the pace of usability work was quite slow, it is acceptable because discussion about one issue takes about 30 minutes and the discussions about such matters sensitizes the developers to GUI design and usability principle issues. The developers have since then expressed their desire for integrating Kitchen Sink closer to the code which has resulted in two projects for such purposes.

2.3.3 UKKOSS research programme

Rajanen and Iivari (2019) have conducted a research programme called UKKOSS, which main aim is to investigate and theorise ways how usability practitioners can participate in OSS development. The research is based on analysis of cases, where student teams acting as usability teams and guided by the researchers, perform usability interventions on different kinds of OSS projects by using different strategies and methods.

Measuring the effectiveness of different approaches to introducing usability activities to OSS

Rajanen et al. (2011; 2012) have experimented with different kind of ways of introducing usability activities to OSS and measured their effectiveness by studying the results of UKKOSS cases. Cases, where the usability team used a consultative approach where they submitted their usability work in the same way as patches are usually submitted to OSS and without prior interaction with the developers, did not end up with impactful changes to the software's usability (Rajanen et al., 2011). Cases that used a participative approach generally yielded better results. In a participative approach, the usability teams aim to become recognised members of the project's community. (Rajanen et al., 2011; Rajanen et al., 2012.) This can be achieved by adapting the usability work to the culture of the specific project and submitting code patches (Rajanen et al., 2012). Rajanen et al. (2011) also suggested several other core components for bringing usability work to OSS successfully, such as understanding the characteristics of OSS development, aiming to make the core developers allies by communicating with them, promoting the interests of end-users, identifying the benefits of improved usability and advocating usability work by referring to them, and adapting the usability work to the development while maintaining an objective view.

Mixing HCI and OSS philosophies

Rajanen and Iivari (2013) examined if the core philosophies of OSS and HCI can co-exist in OSS projects by analysing two UKKOSS cases. In both cases, the usability teams managed to embrace both of the philosophies simultaneously by adjusting their work based on the OSS philosophy (interacting with the community, gaining merit, reporting and fixing bugs, etc.) while adhering to the HCI philosophy by representing the users and influencing the design process of the software in both consultative and participative roles. The authors identified a possible risk of too close involvement in the development process, as according to HCI literature, too close involvement in development may hinder the usability work, as it can be difficult to represent the needs of users if one is too involved in the design process. In one of the cases involving an OSS game, a member of the usability team got quite closely involved in the development as he managed to become a developer due to his contributions to the project, community activities, and his skills as a player of the game. He ended up spending more time on coding than on usability work, but it did not end up becoming a problem, because the other usability team members were able to focus on usability.

Enculturation & OSS project culture types

Iivari et al. (2014) examined UKKOSS cases and a few other usability interventions from the perspective of enculturation. Some of the cases had corporate involvement. The results of the cases indicated that enculturation efforts were beneficial to making an impact with the usability activities. They defined enculturation in this context as the usability teams gaining enough knowledge of the culture they have entered so they can adjust their work accordingly. They argued that enculturation happens naturally when if the usability practitioners are involved in the OSS project from its inception. In two of the seven cases, usability practitioners were involved in defining the OSS project and knew personally the developers. Both of those cases ended up with impactful usability contributions. These kind of cases where usability practitioners are involved right from the start are rare, though. In general, cases with enculturation efforts were more likely to end up with impactful usability contributions. According to the authors, enculturation in OSS consists of understanding the product, motivating usability, and targeting the decision-makers. They recommend usability practitioners to examine the available online material of the OSS project they aim to join and observe the communication channel for a while in order to gain cultural knowledge on the context they are entering into.

Rajanen and Iivari (2015b) investigated the effect of the culture type of an OSS project on the success of usability teams' usability interventions. They derived 4 culture types from the competing values model that is used to categorise cultures based on organisations' value orientations, developed by Denison and Spreitzer (1991, as cited in Rajanen & Iivari, 2015b). The model consists of two axes based on value orientations, diverging from change to stability, and from internal focus to external focus. Change highlights flexibility and stability emphasises control and order. Internal focus emphasises maintenance of the existing system, and external focus emphasises interaction and competing with the organisational environment. (Denison & Spreitzer, 1991, as cited in Rajanen & Iivari, 2015b.) The derived culture types based on the model were:

- Group culture type (change and internal focus)
- Adhocracy culture type (change and external focus)
- Hierarchical culture type (control and internal focus)
- Rational culture type (control and external orientation)

The authors examined four UKKOSS cases, in which two OSS projects had the characteristics of adhocracy culture type, while the remaining ones resembled the group type and the hierarchical type. The usability intervention succeeded only in cases where the OSS project's culture type was adhocracy. The authors suggested that this can be interpreted in a way that adhocracy culture type is the most fitting culture type for usability work, or that the planned usability work on an OSS project has to be modified so it fits the specific culture type. (Rajanen & Iivari, 2015b.)

Power dynamics and gatekeeping tactics

Rajanen and Iivari (2015a) examined 5 UKKOSS cases through the lens of power dynamics, focusing on how the OSS developers wielded power over the usability teams and denied their empowerment. They used the model of power and empowerment developed by Hardy and Leiba-O´Sullivan (1998, as cited in Rajanen & Iivari, 2015a) as a base for their research. The model consists of four dimensions. In the first one, the power is used by managing the dependencies of resources, in the second one by managing decision making processes, in the third one by managing meaning, and in the fourth, the power is embedded into the system itself. All the dimensions are divided into four parts that describe the power dynamics between ones who have power (A) and the ones who do not (B): the first one depicts how A exercises power over B, the second is about the interaction between the groups, the third one describes the reasons why B fails to influence the outcomes, and the last one explains the requirements for the empowerment of B. (Hardy & Leiba-O´Sullivan, 1998, as cited in Rajanen & Iivari, 2015a.) The authors examined how this model applied to the UKKOSS cases, mainly focusing on the second degree of power and empowerment (managing the decision-making processes), though other dimensions were also examined. They defined the OSS developers as the ones with power and the usability teams as the ones without it. They argued that the OSS developers managed the decision making processes by having the access to the decision-making arena and having an influence there. The interaction between the developers and the usability teams consisted of open or covert conflicts, such as developers rejecting the usability teams' contributions or accepting them and reverting them later, or not being willing to communicate about usability issues. As for the reason why usability teams failed to influence the outcomes, the authors argued that usability teams were aware of their position and sometimes able to contact the developers with power, but unable to have an impact on the outcomes. As for the requirements for empowerment, the authors suggested that the usability teams must have influence and gain access to the decision-making arena by either gaining commit rights to the project, or contacting the developers with power and convincing them about the value of their work. (Rajanen & Iivari, 2015a.)

Rajanen et al. (2015) identified three gatekeeping tactics that were used to hinder the usability teams' work in various UKKOSS cases:

- Non-response
- Social exclusion
- False acceptance

Non-response refers to developers not responding to either the messages or contributions of the usability practitioners. It was encountered in a form or another in three of the six examined cases. In one case, the usability team submitted the results of their usability work to the developers via e-mail and later by posting it on the discussion forum of the project, because they did not get a reply to the e-mail. A developer finally commented on the forum that they were discussing with the core developers about the results of the

usability work and that they would comment on it later, but no reply was received. Social exclusion refers to excluding the usability practitioners from the decision-making process of the project. It was encountered in three of the six cases. An example of such tactic happened in one case, where the usability team's work was welcomed by the developers, but the developers considered only fixing the kinds of identified usability issues that they saw as problems. The usability team was also treated like an external resource instead of becoming a part of the project, as the developers did not try to integrate the usability work in the development plan. False acceptance refers to situations where the developers initially accept the usability work but revert the changes later. It was encountered in one of the cases.

3. Research methods

This chapter discusses the context of the research, the research question, and the process of case study research method and how it is applied in this research.

3.1 UKKOSS research programme as the basis of research

This research uses the documented cases of usability interventions conducted to OSS projects as a part of UKKOSS research programme as its basis. Details about the existing research on the UKKOSS cases can be found in the chapter 2.3.3.

UKKOSS research programme's main objective is to find and try new ways how usability experts can meaningfully contribute to OSS projects. It consists of experiments, where student teams acting as usability practitioners enter an OSS project offering their expertise for usability improvements while collecting empirical data for further research. The student teams have been guided by researchers to try out different kinds of approaching strategies to diverse OSS projects. The size of the student teams has varied from 3 to 5, and all of them were required to work 200 to 300 hours on the project. Their tasks consisted of designing and executing the usability improvement activities, measuring the impacts of their usability work, collecting empirical research data about the cases, and writing reports. The students participating in the usability teams have completed at least two courses on usability focused on usability evaluation methods, user interface design and user-centered design. The research programme has been active for over a decade. (Rajanen & Iivari, 2019.)

3.2 Research question

This research is a multiple case study analysing four UKKOSS usability interventions. The main research question is: *"How did the open source software communities react to usability improvement activities conducted by external usability practitioners?"*. The aim is to find out how did the OSS project communities accept the external usability teams' usability activities, how the outcomes of these cases tie into previous research, and to examine what kind of factors may have contributed to the outcomes of the cases through cross-case analysis.

3.3 Case study process

Case study is used as the research method in this research. Runeson and Höst (2009) describe case study research method as a process consisting of four phases: design and planning, collecting data, analysis, and reporting. Case study as a research method allows a researcher to investigate a real contemporary phenomenon by analysing a number of events, conditions and their relationships (Zainal, 2007).

3.3.1 Design and planning

It has been suggested that a researcher planning to conduct a case study should define the rationale for using it as a research method instead of other possible methods, and conduct a comprehensive literature review on the subject (Yin, 2009, p. 2). Case study has been

argued to be a fitting research method when the research question is in the form of "how" or "why" (Yin, 2009, pp. 2-62), when the analysis does not require the control of the examined behavioural events (Benbasat et al., 1987, as cited in Gagnon, 2010, p.16; Yin, 2009, pp. 2-62), when the study is focused on contemporary events (Benbasat et al., 1987, as cited in Gagnon, 2010, p.16; Runeson & Höst, 2009; Yin, 2009, pp. 2-62), and when the examined phenomenon has an established theoretical base (Benbasat et al., 1987, as cited in Gagnon, 2010, p.16). Runeson and Höst (2009) argued that case study as a suitable research method for software engineering research.

Yin (2009, pp. 2-62) recommends using a multiple case design instead of a single case if possible, because it is more likely to produce good results, and the analytic benefits of having multiple cases can be major. Eisenhardt (1989, as cited in Gagnon, 2010, p. 77) explains that cross-case analysis can be used for identifying emerging patterns from the cases by focusing on the similarities and differences between the cases. Light (1979, as cited in Gagnon, 2010, p. 41) argues that the main purposes of studying multiple cases are to create an extensive description of the context of the observed events and to reveal the underlying structure of social behaviour.

Yin (2009, p. 2) argues that a researcher attempting to conduct a case study should also aim to understand the strengths and weaknesses of the research method. According to Zainal (2007), the strengths of case study research method include aspects, such as the great variance of approaches to case studies makes it possible to utilise both qualitative and quantitative data, and the detailed qualitative data that is often gathered in case studies can help to explain possible complexities of real-life settings that would have been probably ignored in other type of research. Yin (2009, pp. 14-15) suggests a perceived weakness of the case study method by arguing that the case study research method has been criticised of lacking rigor, because case study researchers have often allowed things like biased views or ambiguous evidence to have an effect on the conclusions of the study. The generalizability of the case study method has been also questioned due to its dependency on exploring single cases (Tellis, 1997, as cited in Zainal, 2007).

Specifying the cases and the units of analysis have been suggested as some of the main tasks when designing a case study (Runeson & Höst, 2009; Yin, 2009, pp. 24-25). Other suggested tasks include also defining the case study protocol, taking ethical considerations regarding the subjects of the study into account (Runeson & Höst, 2009), developing possible prepositions and theory for the study, and identifying possible issues (Yin, 2009, pp. 24-25). Robson (2002, as cited in Runeson & Höst, 2009) suggests that a case study plan should include the research objective, specifying the case, defining the theory as a frame of reference, explaining the research questions, and defining the data collection methods.

Based on the suggestions in the previous paragraphs of this chapter, a multiple case study is a suitable research method for the selected research question, as it is a "how" type of question, the investigated events are modern, and the research does not require the control of the examined events. A literature review that is used as the frame of reference for the analysis of the cases has been conducted. This research will analyse UKKOSS cases 14-17, which were conducted from 2015 to 2016. The cases were suggested to be researched in this paper and their material was sent to the author by the principal investigator of the UKKOSS programme and the supervisor of this study, Mikko Rajanen. The cases of this case study are the documented usability interventions, and the units of analysis are the attitudes and actions of the OSS developers towards external usability practitioners and their contributions during the usability interventions. Regarding the ethics of the study, the identities of the participants of the usability teams and the OSS developers are not

disclosed. A possible issue in the study is that cases can end up ambiguously regarding the impacts of the usability work due to time constraints of the student projects. This is addressed by testing the latest versions of the selected OSS programs in order to reach more conclusive results regarding the outcomes of the cases and to make cross-case analysis easier.

3.3.2 Collecting data

Runeson and Höst (2009) emphasise that it is important to utilise several data sources in the study to limit the impact of one interpretation of a single data source, as drawing the same conclusion from multiple pieces of evidence strengthens its validity. They explain that several different kinds of data sources, such as interviews, observations, archival data, and metrics can be used in case studies.

The usability interventions studied in this paper had been already conducted before the work on this thesis started. Various research material was gathered and archived during them. The material includes documents, such as project plans, concluding summary reports, the details of conducted usability activities, communication logs between the usability team and the OSS developers, and reports of background information of the involved OSS communities. The author participated in UKKOSS 17 in 2016. I was a part of a team of 3 students and conducted usability work, gathered case data, and implemented some of the GUI modifications we suggested to the software during the project.

The UKKOSS projects that are examined in this thesis consisted of 10 main tasks:

1. Getting acquainted with previous UKKOSS projects
2. Selecting a fitting OSS project
3. Gathering information about the OSS project
4. Planning
5. Contributing to the OSS project
6. Conducting usability activities
7. Reporting the results of the usability work to the OSS project
8. Implementing at least a part of the proposed changes to the user interface
9. Gathering material related to the prior tasks and writing reflective reports
10. Communicating with the lead researcher of the project and the OSS project's community

First, the student teams read the prior research on the UKKOSS research programme to understand the context of the project. After that, they select a suitable OSS project to enter as an external usability team. Being suitable for this research included criteria such as the software being intended for "normal" users, not too many or too few core developers, and not having a planned release of the next version too soon so that there is enough time for contributions and usability work. The gathered background information about the project included aspects like finding out if prior usability activities have been conducted in the project, prior discussion about usability or usability issues, the knowledge level of usability in the community, finding the main communication channels, identifying potential code contribution options, and investigating the hierarchical structure and the culture type of the community. The planning task consists of writing a project plan that determines what in particular is done (contributions, usability activities, communicating with the community, determining how the data is collected and stored, etc.) and by whom, what is the timeline of the project, and what are its possible risks. In the contributing to

the OSS project task, the students attempt to gain merit and recognition within the OSS community by contributing to the project by for example doing tasks specified in the task lists of the community. The conducting usability activities phase consists of performing usability work like heuristic evaluations, cognitive walkthroughs and user testing according to the project plan. After the usability work is done, its results are reported to the OSS developers and at least some of the proposed changes are implemented by the students. In the end, the team gathers data from the previous tasks and writes reports on things like what was done specifically, the success of the usability activities, describing if the usability activities caused changes to the attitude towards usability in the community, and assessing if the activities and the user interface changes caused a lasting effect. (UKKOSS 14 Assignment description, 2015; UKKOSS 15 Assignment description, 2015; UKKOSS 16 Assignment description, 2015; UKKOSS 17 Assignment description, 2016.) In practice, the usability teams of different projects used different approaches. For example, some of them did not contribute code before reporting the results of the usability work to the developers or implement some of the suggested changes themselves. (UKKOSS 14 Final report, 2015; UKKOSS 15 Final report, 2015; UKKOSS 16 Final report, 2015; UKKOSS 17 Summary report, 2016.)

3.3.3 Analysis

The main objective of data analysis is deriving conclusions from the used data while keeping a chain of evidence that links the conclusions to the evidence. (Runeson & Höst, 2009.) Gagnon (2010, p. 72) recommends organising and classifying the gathered data so that it will be easier to analyse. According to him, the data that is not related to the objective of the study should be discarded at this point.

According to Yin (1981, as cited in Gagnon, 2010, p. 76), the researcher should attempt to search for patterns emerging from the data, such as evidence from different sources pointing towards similar conclusions. Gagnon (2010, p. 77) recommends that the researcher gets immersed in the gathered data and examines it several times in order to allow connections and the overall picture of the cases to emerge. Techniques like cross-case analysis can be used for identifying patterns between the cases. (Eisenhardt, 1989, as cited in Gagnon, 2010. p.77).

According to Gersick (1988, as cited in Gagnon, 2010, p. 77), creating a detailed descriptions of all the cases is key to producing theoretical intuitions. Gagnon (2010, p. 80) explains that the purpose of case descriptions is organising the evidence into a narrative that supports the emerging patterns from the cases and returning them into their specific context. According to him, this helps in contextualising the results of the analysis and it is useful for guiding the interpretations of the evidence. He emphasises the importance of reporting also the contextual elements along with the events directly related to the phenomenon that is being investigated, and recommends using quotes in order to be faithful to the evidence.

The approach to data analysis in this research is not highly formal, because the research question is descriptive and interpretive in nature rather than based on testing a pre-determined theoretic proposition. The case descriptions are divided into two parts this study. The context and the overview of the cases are first described in their own chapter, and the research question regarding the reactions of the OSS developers to the usability intervention is answered in the chapter after that. The outcomes of the cases are interpreted in the Discussion and implications chapter through the lens of previous research, and possible emerging patterns in the cases are also examined though cross-case

analysis. The larger goal of this study is to gather insights on what can usability practitioners do in order to have their contributions valued by OSS communities by examining the outcomes of these cases. In order to make the analysis process easier, I filtered irrelevant information by creating two folders, one for material related to giving an overview of the cases, and another for material that was related to the reactions of the developers.

3.3.4 Reporting

When it is time to report the results of the study, the intended audience for the study should be defined (Gagnon, 2010, p.97; Yin, 2009, p. 164), the structure of the report should be designed, and its drafts reviewed by others (Yin, 2009, p. 164). Baxter and Jack (2008) argue that it depends on the researcher to report the findings in a format that the reader can understand. According to them, the main goal of the report should be describing the results in a way that makes the reader feel as if they would have been an active participant of the research. Robson (2002, as cited in Runeson & Höst, 2009) argues that the report should communicate the audience what the study was about, present the data in a form that the reader can understand so the derived conclusions are logical, describe the studied case clearly, and explain the derived conclusions and their context.

The intended audience of this study are researchers interested in usability of OSS, usability practitioners who want to get involved in OSS, and people who want to get an overview of the subject of usability in OSS in general. Drafts of this thesis are peer-reviewed by other students so it can be improved based on the feedback. All the case descriptions are linked to the part of the archive data they are based on by referencing.

4. UKKOSS usability intervention cases

This chapter describes the usability intervention cases based on the data gathered during UKKOSS projects 14-17. It gives an overview of each of the cases by explaining the characteristics of the selected OSS project and describing what kind of work the usability team did during the usability intervention. Some additional background information, such as the licenses of the OSS projects and their download statistics was also gathered in order to provide more detailed descriptions of the cases.

4.1 Case 1: UKKOSS 14 (Mumble)

UKKOSS 14 was conducted in the spring of 2015. The student usability team consisted of four members. They chose Mumble, a voice chat application, as the OSS project community where they entered as external usability practitioners. (UKKOSS 14 Final report, 2015.) The latest version of the software had been downloaded 178195 times in a month (UKKOSS 14 Mumble info report, 2015). The first version of the software was released in 2005, and the project uses BSD license (SourceForge, n.d.-a). During the usability intervention, the team investigated the characteristics of the community, contributed to the Finnish translation of the software, conducted usability work on the project, and gathered data about how the community accepted the usability work for further analysis. (UKKOSS 14 Final report, 2015.)

The usability team investigated the discussion channels of the OSS project before entering. The discussion forum of Mumble had at the time 2446 registered users, of which eight were admins who were core developers of the software. (UKKOSS 14 Community analysis report, 2015.) There were a total of 5208 posted messages and 1460 topics on the forum (UKKOSS 14 Mumble info report, 2015). The admins were seen participating in the discussions actively. The original lead developer who had contributed the most lines of code in the community had left the project in 2012. The usability team described the community as approachable and open based on their observations, and the discussion on the forum as mainly focused on technical issues. Discussion revolved around things like reporting technical issues, asking for help for using the software, and programming new features. They did not find arguments between the members of the forum, and they got an impression that new users were not judged harshly by the community, as experienced users were seen helping new users at novice level issues. (UKKOSS 14 Community analysis report, 2015.)

The team conducted cognitive walkthrough, heuristic evaluation and user testing on the software. The user testing was conducted in the usability laboratory of University of Oulu, with the help of 11 testers whose testing sessions were recorded. The testers had no previous experience with Mumble. A usability report which contained 26 different usability problems was produced and sent to the developers. (UKKOSS 14 Final report, 2015.) The report included prototype pictures of the proposed solutions (UKKOSS 14 Usability report, 2015).

4.2 Case 2: UKKOSS 15 (Task Coach)

UKKOSS 15 was conducted in the spring of 2015 by a team of four students. They chose a time management software called Task Coach as their OSS project. A new version of the software had been recently released and the mailing list of the project was very active.

(UKKOSS 15 Project seminar report, 2015.) In the spring of 2015, the combined amount of downloads of all the files of all releases of the software on SourceForge were over four million (SourceForge, n.d.-c.). The project was started in 2005, and it uses GPL version 3.0 (SourceForge, n.d.-b). It is governed by a small group of core developers (Task Coach, n.d.). During the intervention, the usability team carried out usability work, worked on the Finnish translation of the software, and fixed some of the identified usability issues (UKKOSS 15 Development report, 2015; UKKOSS 15 Project seminar report, 2015).

The usability team conducted heuristic evaluation to the software and used its findings in determining the shortcomings of usability that could be the focus of user testing. The members of the usability team conducted heuristic analysis separately and combined their results in the end. (UKKOSS 15 Project seminar report, 2015.) They carried out usability testing with 14 test users in the usability laboratory of University of Oulu (UKKOSS 15 Final report, 2015). The user tests were focused mostly on new users attempting to learn how to use the software (UKKOSS 15 Project seminar report, 2015). A usability report which included proposed solutions for the identified usability issues was produced and sent to the developers (UKKOSS 15 Final report, 2015). It included also summaries of testers' backgrounds (UKKOSS 15 Usability report, 2015). Two prototype pictures for proposed usability issue solutions were produced (UKKOSS 15 Prototypes, 2015), but they were not included in the usability report (UKKOSS 15 Usability report, 2015). The documents do not address if they were sent to the developers separately (UKKOSS 15 Final report, 2015; UKKOSS 15 Project plan, 2015; UKKOSS 15 Usability report, 2015). The project plan mentions initial plans of using paper prototypes in user testing (UKKOSS 15 Project plan, 2015).

The development of Task Coach started to slow down during the project due to the developers going on hiatus (UKKOSS 15 Project seminar report, 2015). This caused communication problems between developers and the usability team, as the developers stopped replying to the messages the usability team sent (UKKOSS 15 Final report, 2015).

4.3 Case 3: UKKOSS 16 (HandBrake)

UKKOSS 16 was conducted in the spring of 2015. The usability team consisted of four students. They selected HandBrake, a video encoding software for their OSS project. (UKKOSS 16 Final report, 2015.) The combined amount of downloads of all files of all release versions of the software in GitHub amount to nearly seven million as of May of 2021 (GitHub Release Viewer, n.d.-a). The project was started in 2003 (HandBrake Documentation, n.d.), and most of its code is covered by GPL version 2, though some parts of it use BSD 3-clause license (HandBrake, n.d.). During the project, the usability team gathered data about the OSS community, carried out usability work, and sent a usability report to the developers (UKKOSS 16 Final report, 2015).

The usability team examined the OSS project's community. According to them, there were not any usability experts involved in the project, and usability was not regarded as a high priority in the community. The developers seemed to be more interested in adding features to the program. The usability team perceived the project to be run by a small group of hobbyists, and the core developers did not seem to be interested in growing the software's user base. They developers seemed to be focused on expert users instead of tailoring the software for novice users. The usability team observed that although the developers did not seem to be interested in expanding the user base, they rarely outright slammed the ideas of users. The developers usually at least explained the rationale for

rejecting ideas. Some other users of the forum were seen rejecting suggestions rudely without explaining their rationale, though. All in all, the usability team explained that the community seemed to be a quite difficult one to get to accept usability improvement suggestions. (UKKOSS 16 Data report, 2015.)

The usability team conducted cognitive walkthrough, heuristic evaluation and user testing (UKKOSS 16 Data report, 2015), wrote a usability report which included some proposed solutions to the found usability problems (UKKOSS 16 Usability test report, 2015), and sent it to the developers (UKKOSS 16 Final report, 2015). The report included prototype pictures for the proposed solutions and summaries of background information of the testers. The testers were also interviewed. The rationale for evaluating the severity usability issues was explained by using user testing metrics such as the recorded duration of the tasks. (UKKOSS 16 Usability report, 2015.) After sending the usability report, they asked a developer some questions about his opinion on the report and usability of HandBrake in general. (UKKOSS 16 Data report, 2015).

4.4 Case 4: UKKOSS 17 (Streama)

UKKOSS 17 was conducted in the spring of 2016 by a group of three students. The usability team selected Streama, a video streaming application for the user's own videos, for their usability intervention OSS project. The development of the software started in 2015. (UKKOSS 17 Summary report, 2016.) The combined download count of all the files of all the releases of the software was around 66000 in May of 2021 (GitHub Release Viewer, n.d.-b). It uses MIT license (GitHub, n.d.-a). During the intervention, the usability team gathered data about the project, made small code contributions to the software, carried out usability work, and implemented some of their suggested user interface modifications (UKKOSS 17 Summary report, 2016).

The issues page of the project's GitHub site was the only discussion forum of the project. The usability team did not see a record of previous usability activities in this project, and there was not much discussion about usability issues in general. (UKKOSS 17 Streama report, 2016.) The project was governed by a lead developer who decides what changes will be implemented to the software, but listens actively change suggestions of other users (UKKOSS 17 Summary report, 2016).

The usability team conducted heuristic evaluation, cognitive walkthrough and user testing, produced usability reports of each of the methods, and presented them to the lead developer of the project. The reports included suggested solutions for the usability problems, and the lead developer was asked to go through them all and either approve or reject them. (UKKOSS 17 Summary report, 2016.) The proposed solutions did not include prototypes or pictures, and the user testing report did not include specific metrics such as recorded task durations (UKKOSS 17 Cognitive walkthrough report, 2016; UKKOSS 17 Heuristic evaluation report, 2016; UKKOSS 17 User test report, 2016).

4.5 Summary

The characteristics of the OSS projects in these cases were quite similar. Table 1 describes the details of the OSS projects. Table 2 summarises the details of the student teams and their work.

Table 1. Characteristics of the selected OSS projects.

	Mumble (UKKOSS 14)	**Task Coach (UKKOSS 15)**	**HandBrake (UKKOSS 16)**	**Streama (UKKOSS 17)**
Application type	Voice chat	Time management	Video encoding	Video streaming
Starting year	2005	2005	2003	2015
Governed by	A small group of core developers	A small group of core developers	A small group of core developers	A lead developer
Userbase	Medium	Small	Medium	Small
License	BSD	GPLv3	Most of the code uses GPLv2, some parts use BSD 3-clause license	MIT license

As Table 1 shows, the application type of the software varied in the four cases. Most of them were governed by a small group of core developers. The userbase of the projects ranged from medium to small, though it can be difficult to estimate the real size of the projects, because not all of them provide official download statistics and the downloads can be distributed to multiple websites. Each of the OSS projects used a different license.

Table 2. A summary of the UKKOSS projects 14-17.

	UKKOSS 14 (Mumble)	**UKKOSS 15 (Task Coach)**	**UKKOSS 16 (HandBrake)**	**UKKOSS 17 (Streama)**
UKKOSS project year	2015	2015	2015	2016
Usability team members	4	4	4	3
Usability activities	Heuristic evaluation, cognitive walkthrough and user testing	User testing and heuristic evaluation	Heuristic evaluation, cognitive walkthrough and user testing	Heuristic evaluation, cognitive walkthrough and user testing
Usability report's contents	Several usability issues and their proposed solutions with prototype pictures	Several usability issues, their proposed solutions and summaries of testers' backgrounds	Several usability issues and their proposed solutions with prototype pictures, summaries of testers' backgrounds, interviews of test users, and thorough metrics of user testing, such as average duration of tasks and their success rates	Several usability issues and their proposed solutions without prototype pictures

Table 2 visualises how similar the student projects were in terms of what kind of usability work they conducted and how many students were involved. All of the projects were conducted in 2015 and 2016, and most of the student teams consisted of four members. All the teams except UKKOSS 15 (Task Coach) conducted heuristic evaluation, cognitive walkthrough and user testing. In UKKOSS 15, the team did not conduct cognitive walkthrough. The contents of the usability reports of the cases varied. UKKOSS 16

(HandBrake) had the most comprehensive report compared to the others, because it included details such as the recorded duration of the tasks in user testing and data-based reasoning for the evaluation of the severity of the identified usability issues.

5. Findings

This chapter explains how the usability teams approached the OSS communities during UKKOSS projects, describes the reactions of the OSS developers to the usability interventions, and examines the impacts the interventions had on the software's usability.

5.1 How did the open source software communities react to usability improvement activities conducted by external usability practitioners?

The following subchapters describe the reactions of the developers to the usability interventions by going through all the cases individually. The approaches to the usability interventions are also described in detail in order to explain the context of the developers' reactions. In order to reduce the ambiguity of the impacts of the usability interventions, I tested the latest stable versions of the selected OSS programs and evaluated if usability improvements proposed by the usability teams had been implemented later by the developers. The outcomes of all of the cases are summarised and visualised by a table in the end.

5.1.1 Reactions to UKKOSS 14 (Mumble)

In order to get recognition in the community, the usability team of UKKOSS 14 decided to enter the project by implementing Finnish translations for Mumble before moving to usability work. They concluded that the lack of usability discussion on the forum of the project indicated that it would not be a suitable place for contacting the developers. They decided to use the Transifex translation tool which they used for making the Finnish translation for communicating with the developers instead, because they noticed during the translation process that core developers of the OSS project also used it. (UKKOSS 14 Final report, 2015.)

When the translation was nearly finished, the usability team contacted one of the developers and asked him how to get it added to the software. They did not reveal the context of this work as a part of usability research. The developer responded by thanking them for the work: "Thank you for your efforts! We regularly update our development snapshots with the updated translations, so you don't have to do anything to get them included." (UKKOSS 14 Transifex messages, 2015.)

The usability team tried to provoke discussion about usability by mentioning that some of their friends had started using Mumble but encountered usability problems when they sent another message asking help for translation problems. The developer ignored the usability part of the message. The usability team decided to change their approach to a more direct one and they told the developers that they were students on a usability testing course who wanted to conduct usability work on Mumble. (UKKOSS 14 Transifex messages, 2015.) The developer welcomed usability work and was interested in making the software more accessible:

> Of course we would like Mumble to be accessible to everyone. (And of course, because of time, we don't really get to tackeling [*sic*] this and many other issues.) Listing concrete issues, and providing suggestions for improvement would be most likely to change anything. (UKKOSS 14 Transifex messages, 2015.)

The usability team sent a usability report based on the results of their heuristic evaluation, cognitive walkthrough and user testing to the developer. The report included also proposed solutions for the identified usability issues. (UKKOSS 14 Transifex messages, 2015.) The developer accepted the report very enthusiastically, but he was also curious what the results were based on:

> Hey, thank you very much. From the document, it is not clear to me what it is based on / a result of. What kind of testing group was that? Was it supervised, or did some people just list their issues? Did you discuss these in a group? We should definitely put this somewhere. . . . I actually think I'm going to convert the document to a wiki page, so each suggestion can be discussed there. I feel like multiple forum topics would be too hard to follow, and issue tickets - mmmh. Actually, the best would indeed be to open issue tickets for these. Reading through the document, I already got some immediate comments. I also makes me squirrelly, wanting to fix some stuff right away. . . . Again, thank you for your effort! I can definitely agree with a lot of it. Time and motivation is often lacking, so suggestions with clearly prepared solutions are definitely a good thing. (UKKOSS 14 Transifex messages, 2015.)

The usability team asked the developer some questions regarding how he perceived their work, such as what in particular made the usability report good in his opinion, and did the translation work affect the acceptance of the usability report. He responded that he valued identifying concrete issues and their improvement suggestions supplemented with pictures and the fact that it was based on multiple people's work gave it more weight. He also explained that in his opinion, previous contributions to the project give a person more credibility when proposing new ideas. He did not know initially how much translation work the team actually did (which was around 80% of the whole translation), but explained that it did not affect his positive reaction to the report. (UKKOSS 14 Mumble log, 2015.)

The developer informed the usability team later that he had created a wiki page for the new usability findings based on the report and he had commented on the suggested usability issue solutions and asked for additional details about some of the solutions. The usability team responded by explaining the missing details. (UKKOSS 14 Transifex messages, 2015.) The usability team also spent some time chatting with the developer casually (UKKOSS 14 Steam chat log, 2015).

The developer shared the usability report to other core developers. From the total of 26 proposed usability issue solutions, the developers agreed on 13 of them, disagreed with five of them or the solutions were deemed problematic, and left eight of them uncommented. The usability report was left to the developers. They thought of publishing the usability findings to the community to gather comments, but it was unclear who would do so. When the usability intervention ended, the usability team was hopeful that their usability work would get implemented to the later versions of the software, because the usability report was shared among the developers and they commented on the findings. (UKKOSS 14 Final report, 2015.)

In this case, the contact developer reacted very enthusiastically to the usability issue findings and welcomed the translation work. The usability team communicated actively with him during the project, and the chat logs give an impression that the usability team got along well with the developer. Although the initial reaction of the developer was very positive, it seems like not many usability issues described in the report have been fixed. Based on testing the latest stable version of Mumble (1.3.4) which was released in early

2021, Finnish translation has been implemented to the software, but most of the proposed GUI changes have not. I found around ten cases where the developers had agreed with the usability team about specific usability issues but they have not been fixed later. Some changes, such as making the Back buttons more consistent, changing the icon of the configuration menu to a simpler one, changing the icon for audio input testing to support colour blind users, and adding Show command to the system tray menu were implemented as suggested. I could not test if the issue related to Mac OS X was fixed because I do not have access to that operating system and I could not get the error message relevant to one of the issues to appear so I could not evaluate if it was fixed.

5.1.2 Reactions to UKKOSS 15 (Task Coach)

In UKKOSS 15, the usability team initially approached the OSS community by not revealing their identities as usability practitioners, but acting as people who were just interested in usability and the OSS project (UKKOSS 15 Final report, 2015). A usability team member sent an e-mail asking one of the developers about how they felt if he worked on Finnish translation of the software and made changes related to the usability of the software. The developer responded by welcoming him to contribute to the software: "Welcome [usability team member's name], your help is certainly appreciated. I don't think we'll have any particular thoughts until there are specific questions about details." (UKKOSS 15 Introductory e-mail, 2015.)

The team initially planned to integrate into the community by gifting small code patches before starting working on usability (UKKOSS 15 Project seminar report, 2015). They needed to change this approach because they encountered difficulties when setting up the development environment, which made contributing code difficult. The software relied on several third party libraries, which increased the complexity of setting up its development environment. (UKKOSS 15 Final report, 2015.) The team member responsible for implementing the translation and usability work to the software contacted one of the developers about the technical difficulties, and revealed that his work on this OSS project was based on a university course. (UKKOSS 15 Task Coach Setup report, 2015.) The team received help from them, but their assistance was not sufficient according to the usability team, and the usability team's programmer ended up having to solve many of the encountered issues by himself. (UKKOSS 15 Final report, 2015.)

The usability team produced a usability report based on the data gathered during user testing and heuristic evaluation. (UKKOSS 15 Final report, 2015.) The report included 24 proposed solutions for the identified usability issues (UKKOSS 15 Usability test report, 2015). It was sent to the developers (UKKOSS 15 Final report, 2015). The usability team described the reaction of the OSS community to the usability report as welcoming and being impressed of the gathered data. The developers acknowledged some of the found usability problems, and some of the found issues were completely new to them. The usability team and the developers exchanged about six e-mails about the findings. (UKKOSS 15 Project seminar report, 2015.) After that, the developers stopped responding to the messages (UKKOSS 15 Development report, 2015). The usability team sent the delayed Finnish translation patch to the developers at the end of the UKKOSS project (UKKOSS 15 Final report, 2015). The developers did not respond to this contribution (UKKOSS 15 Final report, 2015). The usability team found out that the developers had gone on hiatus with only one of the developers left in the project (UKKOSS 15 Project seminar report).

After the translation was finished, the programmer of the usability team experimented with implementing some GUI changes proposed by the team members responsible for usability work, recorded the changes on video for the other team members, and sent the results to the OSS community (UKKOSS 15 Development report, 2015). The GUI changes consisted of removing menu tabs of features that the usability team evaluated as irrelevant or harmful to usability (UKKOSS 15 Development report, 2015; UKKOSS 15 Suggested UI changes report, 2015; UKKOSS 15 Videos of UI changes, 2015). The usability report suggests removing the task prerequisites tab as a solution to a usability problem (UKKOSS 15 Usability test report, 2015), but the implemented changes removed also two other tabs based on another report which does not mention what usability activities it was based on (UKKOSS 15 Development report, 2015; UKKOSS 15 Suggested UI changes report, 2015; UKKOSS 15 Videos of UI changes, 2015). The developers did not respond to this contribution either (UKKOSS 15 Development report, 2015).

In this case, the developers welcomed the usability report and were impressed by it, but the communication seized when the developers went on hiatus, and they did not respond to the translation and GUI changes contributions. The development of Task Coach has since resumed, and the Finnish translation was added to the software in the version 1.4.3 which was released in the spring of 2016 (Task Coach, 2019). Based on testing the latest version of the software (1.4.6), the developers have not implemented any of the proposed solutions for usability issues made by the usability team and the GUI changes implemented by the usability team were ignored.

5.1.3 Reactions to UKKOSS 16 (HandBrake)

The supervisor of this UKKOSS project asked the usability team to use a stealthy approach in this case, so the team acted as if they were a novice user and did not reveal their background and intent. They started by trying to provoke a discussion about usability by sending a usability issue related message on the forum. The message addressed usability problems encountered by a novice user. (UKKOSS 16 Data report, 2015.) A long time forum user replied defending the usability of the software by referring to its learning curve:

> There is a slight learning curve with any software that performs such complex functions as Handbrake. The tooltips for every parameter are the most thorough and comprehensive I've ever seen! Are your tooltips turned off by chance? That said, code patches are always considered welcome. (UKKOSS 16 Data report, 2015.)

Another new user of the software agreed with the usability team that the program lacked user guidance features, and claimed that old versions of the software had more guidance text that was not hidden or collapsed. (UKKOSS 16 Data report, 2015.) A team member of HandBrake responded to him by denying that the GUI was changed in new versions of the software and defending its usability by stating it was intended for advanced users:

> The GUI design hasn't changed too drastically in years. All the key features are still exposed. Also, bare [*sic*] in mind the target audience for HandBrake is not novice video encoders. . . . If you want an easier interface there are tools that do a far better job than HandBrake but are less powerfully [*sic*] typically. Alternatively if you want more power and want to learn fine details, there are better tools than HandBrake. It's all about finding the tools that are right for you.

> Unfortunately a lot of people want HandBrake to be everything for everyone and that's not a feasible option. We don't want to dumb things down too much, or not over complicate things. It's not an easy balance. That said, at least on the windows side when the windows UI was re-written, not all tooltips came across. If particularly controls are of interest to folk, they can be re-added. (UKKOSS 16 Data report, 2015.)

After that, they created a new account and posted a message which revealed the actual context and the intent their work, but did not mention that the previous account was also theirs. They also sent them a usability report based on the usability work they conducted. This approach was chosen so that they could measure the difference of reactions of the developers to the usability improvement suggestions regarding whether they were posted by a novice or a usability practitioner. The usability suggestions from the both accounts were based on the same data that was gathered during the usability work but presented to the developers in a different way. Their usability work consisted of cognitive walkthrough, heuristic evaluation and user testing. (UKKOSS 16 Data report, 2015.) The usability report included various usability issues of which 14 were detected by heuristic evaluation, 6 by cognitive walkthrough, and the most serious problems encountered during user testing were summarised as seven issues. Twelve proposed solutions to the usability problems were presented of which most were supported by prototype pictures. (UKKOSS 16 Usability test report, 2015.)

A developer responded to the new message by asking if the usability team if they will implement the GUI changes themselves: “Well, you are a dedicated and capable group. In that same sense, is it reasonable to assume that at some point, your section 6 will be reinforced with your own code submissions?” (UKKOSS 16 Data report, 2015.) It is unclear if this forum user was directly involved in the development of HandBrake, but his message count was high. The usability team interpreted this response in a way that the user was hurt by the critique and asking if the usability team will do the suggested changes was a passive aggressive way of implying “do it better yourself if you can”. The usability team waited for four days for other replies, but as it seemed that they will not be more of them, they replied themselves by stating that they do not have the resources to implement the suggested code changes themselves. A moderator responded to that message by explaining that the developers have already implemented some of the suggested changes: “Some of the suggestions in your report have already been implemented. . . . Others are in discussion. . . . Thank you.” (UKKOSS 16 Data report, 2015.)

The usability team confirmed that some GUI changes had been recently implemented to the software that resembled their proposed usability improvements, but they could not determine which ones of them were implemented based on the first message they posted when playing a novice user or the second one where they explained their background. A total of seven usability changes were implemented that seemed to be more or less influenced by the suggestions of the usability team, though it is not clear if some of them were directly influenced by them. (UKKOSS 16 Data report, 2015.) Most of the implemented changes were related to the usability team’s two proposed usability issue solutions about improving the video encoding queue and adding shortcut keys to the software (UKKOSS 16 Data report, 2015; UKKOSS 16 Usability test report, 2015).

Lastly, the team sent a questionnaire to one of the core developers asking about things like did they find the usability report helpful. The developer answered that they found the usability report useful, but recommended that usability practitioners should talk with the developers of the selected OSS project about the intended user base of the software before conducting usability work. He argued that in this case, the usability work was focused on

new users, although in his opinion the main user group is intermediate users with video encoding background. He also commented that normally it would be more appropriate to post things like the usability report on the issue tracker, but since the project did not have one, it was okay to post it on the forum. (UKKOSS 16 Data report, 2015.)

The community's reactions to the usability discussions were mixed, but the developers found the usability report useful and implemented several GUI changes to the software quickly. Based on testing the latest stable version of the software (1.3.3), six GUI changes that resemble the solutions proposed by the usability team have been implemented at some point. The developers have changed the main menu's "Start" button to "Start Encode", added a language option to the Preferences menu, changed the "Start" button of the queue menu to "Start Queue", made the button that returns a video file from the queue more descriptive by supporting it with text, changed the "Source" button to "Open Source", and added a video import command to the File menu. Also several other issues mentioned in the report without suggested solutions seem to have been addressed in one way or another. For example, the developers have implemented a drag and drop function to uploading videos, the users are restricted from adding a video to a queue before setting its mandatory settings, more tooltips have been added, and the "Import SRT" button was changed to more descriptive "Import Subtitle".

In a summary, there were a total of twelve proposed solutions to usability issues of which two influenced usability changes to the software during the intervention, six that have been implemented roughly the same way as suggested at some point, and three that have not been implemented in any form. Many of the identified usability issues without proposed solutions have been addressed at some point, though it is difficult to estimate how large part of the them have been directly influenced by the usability report.

5.1.4 Reactions to UKKOSS 17 (Streama)

The usability team approached the OSS project by contacting the lead developer via e-mail and revealing their intentions and the context of the usability intervention as a part of a university course. (UKKOSS 17 The first contact report, 2016). The lead developer was a usability consultant (UKKOSS 17 Summary report, 2016). She accepted the usability team's help enthusiastically, and suggested a voice call with the project manager of the usability team (UKKOSS 17 First contact report, 2016).

In the call, they discussed about issues, such as the details of the student course, the intended user group of the software (which she defined as average users instead of technical users), and the experience of the usability team. The lead developer suggested some tasks the usability team could do before starting the usability work, such as code refactoring and unit testing. She also offered help if the usability team encountered problems during their work. She had a very positive attitude towards co-operation and was excited about receiving usability reports even if the usability team would not be able to implement all the proposed solutions to the found usability issues, and wanted the team to post the usability findings on the issue tracker of the project after their work is finished. They agreed that the usability team would send soon pull requests of some miscellaneous work, such as unit tests and shortcut keys for the video player, before sending the usability report. (UKKOSS 17 Call report, 2016.)

The usability team submitted the code patches for shortcut keys for the video player and unit tests. Both of them were merged into the main branch of the software, but met with a different level of approval. The format of the unit test patch was deemed good, but the

shortcut keys patch had some code that was implemented in way that was not based on the used main front-end framework. (UKKOSS 17 Pull requests report, 2016.) The lead developer commented the pull request in the following way:

> Hi and thanks for the pull request! I like the changes you made very much, but I'm worried that the code as getting a little bloated now. I would prefer if all the mousetrap changes were extracted into an angular service instead. Also, code like [code example] should not be used and instead a more 'angular-y' way should be found. However, I'll accept the request and fix the above at a later point, or if you fix it in another pull request. Doesn't matter. (UKKOSS 17 Pull requests report, 2016.)

The usability team produced usability reports with proposed solutions for the found usability issues based on the data gathered during the usability work, which consisted of heuristic evaluation, cognitive walkthrough and user testing (UKKOSS 17 Summary report, 2016). Separate reports were made for the results of each of the usability engineering methods (UKKOSS 17 Cognitive walkthrough report, 2016; UKKOSS 17 Heuristic evaluation report, 2016; UKKOSS 17 User testing report, 2016). The usability team sent the usability reports to the lead developer and asked for feedback (UKKOSS 17 Summary report, 2016).

The usability reports were structured in a way that allowed the developer to approve or reject the suggested changes and to add comments if she had questions about the solution or explain why it was rejected. The documents were well received. (UKKOSS 17 Usability activities report, 2016.) The developer replied:

> Ok went through all of em [*sic*] now and commented and approved and stuff. Except for the user tests one as I felt that there were too many duplicates of previously mentioned tasks. If you find that there are several points in there that I have not addressed please let me know and send me another list of just those points. You guys were super thorough!! Thank you for contributing so much! I hope you will be able to implement some of those suggestions! (UKKOSS 17 Usability activities report, 2016.)

A total of 96 usability issues were found of which 10 were considered serious, and the lead developer accepted the proposed solutions for 58 of them. All of the 20 issues that came up in user testing were ignored due to them being too similar to the issues mentioned in heuristic evaluation and cognitive walkthrough reports. The most common reason for rejecting a suggested change was that the suggested solution was not specific enough. (UKKOSS 17 Summary report, 2016.) Other common reasons for rejection included the suggestion being too difficult to implement technically, or the upcoming new version was supposed to make the issue irrelevant. (UKKOSS 17 Cognitive walkthrough report, 2016; UKKOSS 17 Heuristic evaluation report, 2016).

The usability team implemented the proposed solutions for 38 of the usability issues. The implemented changes included new error and acceptance messages, unifying contradictory texts, making the existing notifications and guide texts more specific, modifying the menu structure, and adding a user guide page. The lead developer thanked the usability team for their contributions and merged the changes to the software. (UKKOSS 17 Summary report, 2016.) Some of the remaining unimplemented accepted solutions to usability issues, miscellaneous usability improvement suggestions and encountered bugs were posted on the issue tracker (GitHub, n.d.-b; UKKOSS 17

Cognitive walkthrough report, 2016; UKKOSS 17 Heuristic evaluation report, 2016; UKKOSS 17 Improvements & bugs report, 2016).

The project manager contacted the lead developer for the last time after the code submissions were accepted and clarified some of the ambiguous proposed usability issue solutions. The lead developer seemed to be happy with the usability work. She commented: "Thank you a bunch! It was fun working with you guys, and I am always happy about useful PRs which yours most definitely were. Thank you again!" (UKKOSS 17 Final comments report, 2016.)

The reactions to the code contributions and usability reports by the developer were mostly very positive, but there were some issues, such as the incompatibility of some of the code with the used front-end framework. Based on testing the latest version of the software (1.10.3) which was released in early 2021, seven of the remaining proposed usability issue solutions approved by the lead developer have been implemented at some point, though it is not clear how many of them were directly influenced by the usability report because some parts of the GUI of the software were being redesigned during the usability intervention. The implemented GUI modifications consisted of removing the side panel from the main page in order to unify the button placements, the invited user's role selection was changed to use checkboxes in the invitation menu, a feature was added that allows the user to write notifications about a specific movie on the notifications page, deleting multiple videos simultaneously was allowed, a completion notification was added to uploading subtitles, the user's role selection menu was changed to use checkboxes, and a completion notification message was added to the video uploading menu. 17 of the 27 issues that were posted on the issue tracker have been fixed in a way or another and closed between 2016 and 2018 (GitHub, n.d.-b).

5.1.5 Summary

Table 3 summarises the outcomes of the four usability interventions and how the usability teams approached their OSS projects.

Table 3. A summary of the outcomes of UKKOSS projects 14-17.

	UKKOSS 14 (Mumble)	**UKKOSS 15 (Task Coach)**	**UKKOSS 16 (HandBrake)**	**UKKOSS 17 (Streama)**
Usability team's approach to the usability intervention	Worked on Finnish translation and discussed with a developer before moving to usability work	Initially planned to act as users who were just interested in the software, but revealed their background later	Started conversation about usability issues without revealing their intent and later created a new account that revealed their intent and presented their usability work	Contacted the lead developer and discussed with her about which ones of the suggested usability issue solutions were acceptable
Contributions not based on the usability report	Finnish translation	Finnish translation	None	Unit tests and shortcut keys to the video player
The usability team fixed some of the usability issues	No	Yes	No	Yes
The reactions of the OSS developers to the usability intervention	The developers welcomed translation work and accepted the usability report very enthusiastically	The developers welcomed usability work and were impressed by the usability report, but the communication between the usability team and the developers eventually stopped when the developers went on hiatus	Mixed reactions to the initial usability discussions, but the developers found the usability report helpful and started on working on some the identified usability issues without notifying the usability team	The lead developer accepted the usability work enthusiastically
Proposed solutions for identified usability issues	26 issues of which the developers agreed on 13 of them, disagreed on 5 of them, and left 8 of them uncommented	The developers acknowledged some of the 24 identified issues and found some of them completely new	The developers made several GUI changes based on two of the twelve proposed solutions during the intervention	96 issues of which 58 of them were accepted by the lead developer
Impacts of usability intervention	The Finnish translation was implemented and four of the identified usability issues have been fixed as suggested by the team after the intervention	The Finnish translation was accepted to a subsequent version of the software but the proposed solutions to usability problems have been ignored by the developers	The developers implemented quickly GUI changes influenced by the usability team and several of the remaining usability issues have been addressed at some point	The 38 GUI changes implemented by the usability team were accepted, and seven of the remaining approved usability issue solutions have been implemented at some point

The contributions of the usability team to the OSS project and the approach to the usability intervention varied between the cases. Although the developers accepted the

usability work generally gladly, the amount of the implemented usability improvements ranged from none to many between the cases.

6. Discussion & implications

The developers reacted to the usability interventions generally in a positive way and accepted the usability work enthusiastically in most of the cases, but the impacts of the usability work varied greatly. UKKOSS 16 (HandBrake) was the only case where the community had initially mixed reactions to the usability improvement discussions, but the developers seemed to appreciate the produced usability report based on the interview the usability team conducted. Three of the four cases ended up with at least some GUI changes influenced by the usability reports to the software. In UKKOSS 14 (Mumble), the developers have fixed some of the found usability problems after the intervention. UKKOSS 15 (Task Coach) was the only case where the developers did not end up implementing any of the solutions proposed by the usability team. In this case, the communication between the usability team and the developers stopped abruptly when the developers went on hiatus. In the case of UKKOSS 16 (HandBrake), the developers started addressing usability issues immediately after they were notified of them, and they have also implemented various changes to the software that address the issues mentioned in the usability report. In UKKOSS 17 (Streama), the usability team implemented many of their proposed solutions to the identified usability problems themselves, and the lead developer accepted their code contributions. After the usability intervention ended, seven of the remaining accepted proposed usability issue solutions have been implemented at some point. The gift contributions unrelated to the identified usability issues, such as the Finnish translations and unit tests were accepted by the developers in all of the three cases where the usability team made them.

Prior research suggests that usability practitioners often have difficulties to get their voice heard when attempting to contribute usability work in OSS projects (Çetin et al., 2007; Rajanen et al., 2015; Rajanen & Iivari, 2015a). This can be seen also in most of the cases analysed in this research. Cases, where the developers were initially very enthusiastic towards the usability work but only a few or none proposed GUI changes ended up in the software were common. This can be seen especially in UKKOSS 15 (Task Coach), where the developers were impressed by the gathered usability data and acknowledged some of the usability problems, but they did not implement any of the suggested solutions for the usability problems. In UKKOSS 14 (Mumble), the contact developer was very excited about the usability report and agreed on many of the found usability problems, but only four of the suggested usability fixes ended up in the software. He suggested that the lack of time and motivation affect the willingness to implement usability work. Motivation issues of developers (Andreasen et al., 2006; Nichols & Twidale, 2003) and the lack of resources to fix usability issues have been also acknowledged in prior research (Zhao & Deek, 2005). In UKKOSS 17 (Streama), the usability team implemented many of the GUI changes they suggested after discussing them with the lead developer, and they were accepted to the software. Implementing the proposed changes bypasses the motivational barrier of the developers. This approach should be examined further in future research, though it would not probably be an ideal solution. It has been suggested that usability practitioners often lack technical skills (Nichols & Twidale, 2003). Some GUI changes were also implemented by the usability team of UKKOSS 15 (Task Coach). The developers did not respond to the contribution, but the outcome of that case is difficult to analyse due to the core developers going on hiatus during the project. Another thing to note in that case is that the code contributions focused on removing features of the program. It is understandable from the developers' perspective to not be happy about a contribution that removes parts of the software, even though it would make sense from usability perspective. It is not also clear if the usability team discussed about their changes

with the developers beforehand, and what usability activities were most of the GUI changes based on. The outcome of UKKOSS 17 (Streama) suggests that getting the approval of the developers for implementing GUI changes before actually implementing them could be important.

In UKKOSS 14 (Mumble), a developer commented that the developers of the project often lack time and motivation to focus on usability issues of the software, and recommended usability practitioners to present clear proposed solutions to concrete usability problems in order to increase chance of changing the GUI of the software. The developer was also interested in what data were the suggested changes were specifically based on. Although not many proposed usability improvements ended up in the software in this case, it sounds reasonable to write usability reports in a way that explains what to do specifically and display the metrics that were used during usability engineering in order to minimise the implementation effort of developers and to provide proof of the relevance of the found usability issues thus making the report more persuasive. It has been suggested that it can be problematic that usability issues can be perceived as subjective (Nichols & Twidale, 2006). This problem could be mitigated by providing detailed user testing data. Also if the usability practitioners leave from the project before their usability improvement suggestions are implemented, it may be useful for the document to be as explanatory and independent as possible. In UKKOSS 16 (HandBrake) the developers ended up implementing several changes to the GUI even though the usability team chose an approach where they did not try to gain merit in the community by contributing to the project beforehand. They just submitted the usability report on the forum. A more participative approach has been recommended by prior research on OSS usability interventions (Rajanen et al., 2012). The usability report made by this team was very detailed regarding the used metrics and methods and it included proposed solutions to issues with prototype pictures. In the report, the severity of the usability issues was evaluated by data such as task completion time and task success rate. The usability report of the only case without any usability changes besides the Finnish translation, UKKOSS 15 (Task Coach), was quite plain. It did not include specific metrics of user testing, and the proposed GUI changes were not supplemented with prototypes. Two prototype pictures were produced, but it is not clear if they were sent to the developers or just used in user testing. Some of the suggested solutions were also quite ambiguous. The usability reports of UKKOSS 17 (Streama) also lacked metrics and prototypes, but the lead developer accepted most of the suggested solutions to usability problems. The facts that the lead developer of that case was interested in usability and the usability team members implemented many of the suggested solutions themselves may have mitigated the issue of a plain usability report.

Prior research suggests that convincing the decision-makers of an OSS project about the importance of usability is and making them allies is important for increasing the influence usability work has on the project (Rajanen et al., 2011; Rajanen & Iivari, 2015a; Rajanen & Iivari, 2019). Çetin et al. (2007) suggested that teaching OSS developers basic usability principles would increase the mutual understanding between the developers and the usability experts. The positive outcome of the UKKOSS 17 (Streama) case, where the most of the suggested usability changes ended up being implemented to the software can be interpreted to be greatly influenced by the fact that the lead developer was interested in usability, as she was a usability consultant. Communication about usability problems was easy due to similar knowledge base, and there was no need to try to convince her about the benefits of usability work. This case had also other characteristics that prior research has found beneficial to making an impact on the usability of the software. It has been suggested that earlier the usability practitioners enter the project, the better their chances of having an influence on the usability of the software are (Çetin et al., 2006, as

cited in Çetin et al., 2007; Çetin et al., 2007), and the need to convince multiple people about the benefits of usability can be problematic (Muehling & Reitmayr, 2006, as cited in Çetin et al., 2007). The OSS project of this case was very new, as it was started in the year before the usability intervention, and the lead developer was the only one whose opinion on usability mattered.

The lack of trust between usability practitioners and OSS developers has been discussed in prior research as one of the problems hindering the effective collaboration between them. Face-to-face meetings have been suggested as a one way of establishing trust between usability practitioners and OSS developers (Andreasen et al., 2006.) UKKOSS 17 (Streama) was the only case of the four where the usability team contacted the lead developer via voice call. Even though calling may not be as effective form of communication as meeting face-to-face, it could be a better way to establish trust than asynchronous messaging. This approach could be examined further in future research.

Two of the three gatekeeping tactics identified by Rajanen et al. (2015) were encountered during the usability interventions. It can be argued that social exclusion was used by the core developers to a varying extent in most of the cases because in the end, they were the ones responsible for deciding which identified usability issues they would fix and how, often guided by the logic of what they considered as usability problems or good solutions to them. In UKKOSS 16 (HandBrake), the developers were influenced by the usability report, but they seemed to address the usability issues often on their own terms instead of implementing the suggested solutions directly. It is debatable if the communication problems in UKKOSS 15 (Task Coach) where the developers stopped responding to the usability team's messages can be considered as the non-response gatekeeping tactic, because the developers went on hiatus during the usability intervention. It is unclear if they paused the development because they wanted to reject the usability suggestions, or if hiatus starting during the intervention was purely coincidental. As for the false acceptance gatekeeping tactic, I did not find cases where already implemented GUI changes based on the usability team's work would have been reverted when testing the latest versions of the software, though I was more focused on evaluating if the remaining proposed usability issue solutions had been implemented.

Submitting code has been suggested as beneficial when attempting to become a recognized member of an OSS community (Rajanen et al., 2012) and as a way to gain perceived merit and influence in the community (Gacek & Arief, 2004). Two cases where Finnish translations were given as gifts to the developers (UKKOSS 14 and UKKOSS 15) did not end with many impactful changes to the GUI, although the translation patches were accepted to the software. The developer who was interviewed in UKKOSS 14 (Mumble) explained that the translation work did not affect his positive reaction to the usability report, although he did not know how much work the usability team actually did when translating the software. He said that in his opinion prior contributions give more credibility to new users who are proposing new ideas, though. The impact of gifting the translation patch is difficult to interpret in UKKOSS 15 (Task Coach), because the team sent it at the end of the project due to technical difficulties instead of sending it before the usability report, and the developers also went on hiatus during the intervention. In UKKOSS 17 (Streama), the usability team contributed code gifts such as unit tests and shortcut keys that were suggested by the lead developer, and they were accepted. The impact of the gifts is difficult to measure also in this case, because the case had also other characteristics that were recommended by prior research. Thus the results are inconclusive and further research is recommended.

6.1 Critique of this research

There were many aspects of this research that can be criticised. For example, it was difficult to evaluate if usability changes of the recent versions of the OSS programs were directly influenced by usability reports. Some of the changes addressed issues mentioned in the reports, but often in a different way than suggested by the report. In some cases, general GUI changes may have also rendered the identified usability issues irrelevant, and it is hard to tell if that was the intention or just a coincidence.

Some of the research data was lacking in details of the reactions of the developers, such as direct quotes. This makes the descriptions of the developers' reactions more biased. Some of the reports were also slightly ambiguous regarding what specific documents or contributions were sent to the developers. Ideally all the relevant background information of the OSS projects would have been gathered during the usability interventions, but due to some missing details additional data was gathered during this research, which is not optimal.

The literature review used some sources which were not very scientific. For example, the essays of Eric Raymond were used due to their influence to the OSS literature. The sample sizes of some of the interview-based research papers were quite small, and some of the used research papers were old. The evaluation of the size of the userbase of the OSS projects was also based on questionable metrics. Since most of the analysed OSS projects did not offer official download counts on their main websites, the download counts of alternative download mirrors were used, and their download counts statistics were often in a format that listed the total downloads of all the files of the releases which can be misleading. GitHub did not offer download counts, so an alternative site which credibility can be questioned was used for polling them via GitHub's API.

The developers' hiatus in UKKOSS 15 (Task Coach) makes it difficult to interpret the outcome of the case, because it is not clear if the timing of the hiatus was coincidental or did the developers purposefully avoid communicating with the usability team. It is also possible that instead of rejecting the usability report's suggestions the developers simply forgot it after returning from the hiatus.

The fact that the usability teams consisted of students may have affected the reaction of the OSS developers to the usability interventions, because the developers may have not taken their expertise seriously. Also the student status may have caused the developers to think that the usability team were joining the project only due to the school work instead of being genuinely interested in the software. The short duration of the interventions may have also affected the outcomes. It is not clear if the outcomes of the cases would have been better if the teams would have stayed in contact with the developers for a longer time and discussed more about the usability issues.

7. Conclusions

This research examined how OSS developers reacted to usability improvement activities carried out by external usability practitioners in four usability intervention cases. The developers welcomed usability work more or less enthusiastically in all of the cases, but the impact the interventions had on the usability of the software varied. The outcomes of the cases ranged from none of the suggested solutions ending up in the software to most of them getting implemented. Three of the four cases ended up with at least some of the proposed usability improvements being fixed.

The outcomes of the cases supported the previous research on the difficulty of usability practitioners making impactful usability contributions to OSS, as half of the cases ended up with zero to a few usability improvements being implemented. The other half ended up with more GUI modifications to the software. In the case with the most usability changes to the software, the usability team implemented many of their proposed GUI changes themselves based on the discussions with the lead developer. This case was also the only one where the usability team communicated with the developers via voice call instead of communicating only via asynchronous messaging such as e-mail. The other case with a quite strong influence on the usability of the software did not use a very participative approach which is supported by prior research, but their usability work had an impact anyway. That case had a particularly detailed and persuasive usability report, which established the identified usability problems as concrete issues by displaying the user testing metrics they were based on.

The main contributions of this research are that the results of this study supported the prior research on the issues usability practitioners face when contributing to OSS and new areas of research were proposed that can be explored in future research. Researchers interested in usability of OSS benefit from the results of this study by receiving empirical support for existing theory on the subject of usability practitioners contributing to OSS. For example, the case with the most usability changes had lots of characteristics that were suggested by prior research to be beneficial for impactful usability work. The hardships of external usability practitioners joining OSS projects mentioned in prior research also were encountered in many of these cases. Based on the outcomes of these cases, some ideas were proposed on what kind of approaches to usability interventions could be tested by researchers in future usability studies in order to measure if they would be beneficial to making impactful usability contributions. The mixed results of some of the approaches utilised in the usability interventions, such as usability practitioners implementing their suggested GUI changes themselves, emphasise that there is a need for further research on them. The results of this study can be also beneficial to usability practitioners who want to contribute to OSS projects, because it gives an overview of what kind of obstacles they can encounter so they can prepare to face them. They can also learn from the mistakes and shortcomings of the discussed cases and try out the approaches that worked most effectively.

7.1 Limitations of this study

This study analysed only four usability interventions, so the results are not highly generalizable and the selected OSS projects represent only very small part of OSS. Interpreting the outcomes of the cases and the research material is prone to errors. This research examined only small to medium-sized OSS projects, and none of them had corporations involved in the development.

7.2 Possible future research

More research on the effect of usability practitioners implementing their suggested usability improvements to the impact of their usability work is needed. The case where the usability team implemented many of the suggested GUI changes themselves had also other significant characteristics that have been suggested to be beneficial for making impactful contributions in prior research, so it is difficult to interpret how major role it played in that case. Also the effectiveness of establishing trust between the OSS developers and external usability experts by contacting the developers by calling them or via video conferencing instead of using only asynchronous communication could be examined further.

Future usability intervention studies could also examine how much the depth of the usability report affects the impact the intervention has on the usability of the software by attempting to make the report as persuasive as possible. This could be achieved by including user testing metrics that would provide proof of concrete usability issues and presenting clear solution suggestions with example prototypes in order to reduce the implementation effort of the developers.

It would be also useful to examine the viewpoints of OSS developers themselves to usability interventions in order to understand why usability contributions are often rejected. This could be done for example by conducting follow-up studies on usability interventions where the developers rejected the usability work of external usability practitioners and interviewing the developers.

A developer of UKKOSS 14 (Mumble) mentioned that he would create issue tickets for the identified usability problems, but it is not clear if he eventually did so. Also a developer of UKKOSS 16 (HandBrake) mentioned that he considered the issue tracker to be typically the most appropriate place for posting a usability report. In UKKOSS 17 (Streama), some of the usability issues that were fixed later were posted on the project's issue tracker as the lead developer requested. This approach, where usability issues are handled the same way as functional issues could be examined further in future research. It could be investigated if this would affect implementation rate of the suggested solutions, as the whole community could discuss the issues in the same way that they discuss the technical part of the software. This would also encourage other community members to implement the solutions. This method could be also combined with the previously mentioned emphasis on the persuasiveness of the usability report by including user testing metrics that strengthen the validity of that specific usability issue that is posted on the tracker.

References

Aberdour, M. (2007). Achieving quality in open-source software. *IEEE software*, *24*(1), 58-64.

Andreasen, M. S., Nielsen, H. V., Schrøder, S. O., & Stage, J. (2006). Usability in open source software development: opinions and practice. *Information technology and control*, *35*(3).

Bach, P. M., DeLine, R., & Carroll, J. M. (2009, April). Designers wanted: participation and the user experience in open source software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 985-994).

Baxter, P., & Jack, S. (2008). Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, *13*(4), 544-559.

Benson, C., Muller-Prove, M., & Mzourek, J. (2004, April). Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans. In *CHI'04 extended abstracts on Human factors in computing systems* (pp. 1083-1084).

Bergquist, M., & Ljungberg, J. (2001). The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, *11*(4), 305-320.

Bezroukov, N. (1999). A second look at the Cathedral and the Bazaar. *First Monday, 4(12)*. https://doi.org/10.5210/fm.v4i12.708

Bretthauer, D. (2002). Open source software: a history. *Information Technology and Libraries*, *21*(1), 3-11.

Bødker, M., Nielsen, L., & Orngreen, R. N. (2007, July). Enabling user centered design processes in open source communities. In *International Conference on Usability and Internationalization* (pp. 10-18). Springer, Berlin, Heidelberg.

Çetin, G., Verzulli, D., & Frings, S. (2007, July). An analysis of involvement of HCI experts in distributed software development: practical issues. In *International Conference on Online Communities and Social Computing* (pp. 32-40). Springer, Berlin, Heidelberg.

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004, November). Effective work practices for software engineering: free/libre open source software development. In *Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research* (pp. 18-26).

Dagenais, B., Ossher, H., Bellamy, R. K., Robillard, M. P., & De Vries, J. P. (2010, May). Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* (pp. 275-284).

Ducheneaut, N. (2005). Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, *14*(4), 323-368.

Feller, J. & Fitzgerald, B. (2000). A Framework Analysis of the Open Source Development Paradigm. Proceedings of the twenty first international conference on Information systems, ICIS 2000, Brisbane, Australia, December 10-13, 2000, (pp. 58-69).

Fitzgerald, B. (2006). The transformation of open source software. *MIS quarterly*, 587-598.

Gacek, C., & Arief, B. (2004). The many meanings of open source. *IEEE software*, *21*(1), 34-40.

Gagnon, Y.-C. (2010). *The Case Study As Research Method : A Practical Handbook*. Les Presses de l'Université du Québec.

GitHub. (n.d.-a). *Streama*. Retrieved May 15, 2021, from https://github.com/streamaserver/streama

GitHub. (n.d.-b). *Streama Issues*. Retrieved May 27, 2021, from https://github.com/streamaserver/streama/issues?q=is%3Aissue+involves%3AJanneNiemela+involves%3ALorenzoGarbanzo+involves%3Ajsniemela+created%3A2016

GitHub. (2020, December 2). *The 2020 State of the Octo-Verse*. https://octoverse.github.com

GitHub Release Viewer. (n.d.-a). *HandBrake*. Retrieved May 12, 2021, from https://hanadigital.github.io/grev/?user=HandBrake&repo=HandBrake

GitHub Release Viewer. (n.d.-b). *Streama*. Retrieved May 15, 2021, from https://hanadigital.github.io/grev/?user=streamaserver&repo=streama

GNU. (2015, June 2). *The GNU manifesto.* https://www.gnu.org/gnu/manifesto.html

HandBrake. (n.d.). *Features*. Retrieved May 15, 2021, from https://handbrake.fr/features.php

HandBrake Documentation. (n.d.). *The History of HandBrake*. Retrieved May 15, 2021, from https://handbrake.fr/docs/en/1.3.0/about/history.html

Hars, A., & Ou, S. (2002). Working for free? Motivations for participating in open-source projects. *International journal of electronic commerce*, *6*(3), 25-39.

Hedberg, H., Iivari, N., Rajanen, M., & Harjumaa, L. (2007, May). Assuring quality and usability in open source software development. In *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)* (pp. 2-2). IEEE.

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research policy*, *32*(7), 1159-1177.

Holzinger, A. (2005). Usability engineering methods for software developers. *Communications of the ACM*, *48*(1), 71-74.

Iivari, J., & Iivari, N. (2006, January). Varieties of user-centeredness. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)* (Vol. 8, pp. 176a-176a). IEEE.

Iivari, N., Rajanen, M., & Hedberg, H. (2014). Encouraging for Enculturation–An Enquiry on the Effort of Usability Specialists Entering OSS Projects. ACIS.

International Organization for Standardization. (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models* (ISO/IEC Standard No. 25010:2011). https://www.iso.org/standard/35733.html

International Organization for Standardization. (2018). *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts* (ISO Standard No. 9241-11:2018). https://www.iso.org/standard/63500.html

International Organization for Standardization. (2019). *Ergonomics of human-system interaction — Part 210: Human-centered design for interactive systems* (ISO Standard No. 9241-210:2019). https://www.iso.org/standard/77520.html

Lisowska Masson, A., Amstutz, T., & Lalanne, D. (2017, May). A usability refactoring process for large-scale open source projects: The ILIAS case study. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 1135-1143).

Nichols, D., & Twidale, M. (2003). The Usability of Open Source Software. *First Monday*, *8*(1). https://doi.org/10.5210/fm.v8i1.1018

Nichols, D. M., & Twidale, M. B. (2006). Usability processes in open source projects. *Software Process: Improvement and Practice*, *11*(2), 149-162.

Nielsen, J. (1994a). *Usability engineering*. Morgan Kaufmann.

Nielsen, J. (1994b, April). Usability inspection methods. In *Conference companion on Human factors in computing systems* (pp. 413-414).

Open Source Initiative. (n.d.). *Licenses & Standards.* Retrieved March 12, 2021, from https://opensource.org/licenses

Open Source Initiative. (2007, March 22). *The Open Source Definition*. https://opensource.org/osd

Open Source Initiative. (2018, October 18). *History of the OSI*. https://opensource.org/history

Rajanen, M., & Iivari, N. (2013, October). Open source and human computer interaction philosophies in open source projects: Incompatible or co-existent?. In *Proceedings of International Conference on Making Sense of Converging Media* (pp. 67-74).

Rajanen, M., & Iivari, N. (2015a, April). Power, empowerment and open source usability. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 3413-3422).

Rajanen, M., & Iivari, N. (2015b, May). Examining usability work and culture in OSS. In *IFIP International Conference on Open Source Systems* (pp. 58-67). Springer, Cham.

Rajanen, M., & Iivari, N. (2019). Empowered or disempowered? An analysis of usability practitioners' interventions in open source projects. *Psychological Perspectives on Empowerment, Nova Science Publishers, Hauppauge*, 1-45.

Rajanen, M., Iivari, N., & Anttila, K. (2011). Introducing usability activities into open source software development projects–searching for a suitable approach. *Journal of Information Technology Theory and Application*, *12*(4), 5-26.

Rajanen, M., Iivari, N., & Keskitalo, E. (2012, October). Introducing usability activities into open source software development projects: a participative approach. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Thorough Design* (pp. 683-692).

Rajanen, M., Iivari, N., & Lanamäki, A. (2015, September). Non-response, social exclusion, and false acceptance: Gatekeeping tactics and usability work in free-libre open source software development. In *IFIP Conference on Human-Computer Interaction* (pp. 9-26). Springer, Cham.

Raymond, E. S. (1998). Homesteading the Noosphere. *First Monday*, *3*(10). https://doi.org/10.5210/fm.v3i10.621

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, *12*(3), 23-49.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131-164.

Sears, A. (1997). Heuristic walkthroughs: Finding the problems without the noise. *International Journal of Human-Computer Interaction*, *9*(3), 213-234.

SourceForge. (n.d.-a). *Mumble*. Retrieved May 15, 2021, from https://sourceforge.net/projects/mumble/

SourceForge. (n.d.-b). *Task Coach*. Retrieved May 15, 2021, from https://sourceforge.net/projects/taskcoach/

SourceForge. (n.d.-c). *Task Coach Download Statistics*. Retrieved May 5, 2021, from https://sourceforge.net/projects/taskcoach/files/stats/timeline?dates=2005-02-01+to+2015-05-01

SourceForge. (n.d.-d). *The Complete Open-Source and Business Software Platform*. Retrieved April 5, 2021, from https://sourceforge.net

Steinmacher, I., Silva, M. A. G., Gerosa, M. A., & Redmiles, D. F. (2015). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, *59*, 67-85.

Task Coach. (n.d.). *Task Coach*. Retrieved May 15, 2021, from https://www.taskcoach.org/

Task Coach. (2019, April 27). *Change history*. https://www.taskcoach.org/changes.html

Viorres, N., Xenofon, P., Stavrakis, M., Vlachogiannis, E., Koutsabasis, P., & Darzentas, J. (2007, July). Major HCI challenges for open source software adoption and development. In *International Conference on Online Communities and Social Computing* (pp. 455-464). Springer, Berlin, Heidelberg.

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

Von Krogh, G., & von Hippel, E. (2003). Special issue on open source software development. *Research Policy*, *32(7)*, 1149-1157. https://doi.org/10.1016/S0048-7333(03)00054-4

Väänen-Vainio-Mattila, K. (2011). Käytettävyys ja käyttäjäkeskeinen suunnittelu. In Oulasvirta, A. (Ed.). *Ihmisen ja tietokoneen vuorovaikutus* pp. 102-126. Gaudeamus, Helsinki University Press.

Yin, R. K. (2009). *Case study research: Design and methods* (4th ed.). Sage Publications.

Zainal, Z. (2007). Case study as a research method. *Jurnal kemanusiaan*, *5*(1).

Zhao, L., & Deek, F. P. (2005). Improving open source software usability. *AMCIS 2005 Proceedings*, 430.