Introducing Usability Activities into Open Source Software Development – Impacts of Contributed Usability Evaluations to the Community and Existence of HCI and OSS Philosophies: Case Concrete5

University of Oulu
Department of Information Processing Science
Master's Thesis
Juho-Mikko Balac
15.5.2016

Abstract

Usability is one of the software quality attributes and plays an important role in information systems success, technology acceptance and adoption models. Usability is a product of user interaction design and can be evaluated using usability testing and inspection methods. The usability of open source software (OSS) systems has had a poor reputation. The reasons for poor usability are cultural and software engineers have designed software for their own needs and for tech-savvy users, and thus, usability is not the primary objective. Also, usability experts do not participate much to OSS development. Sometimes they are not welcome or it is difficult to show their merits without contributing code. However, usability of OSS product is getting better as user-centred movement has been closing the gap between users and programmers.

The aim of this thesis is to examine the impacts usability evaluations to an open source software community and to find out the existence of core human-computer interaction and open source software philosophies during the project. The research method used in this study is longitudinal interpretive case study and the research material consists of usability evaluation project material, email correspondence with the community and publicly available information provided by the community and its members.

The attitudes of the target open source software community towards usability and usability activities were mainly positive. After the first contribution, a project leader asked for more activities and after contributing three sets of activities for different versions of the software, the project leader rated them as beneficial. Furthermore, most of the core human computer interaction (HCI) and OSS philosophies existed. The characteristics of HCI philosophy found during the usability project were that usability specialists were user representatives in informative and consultative roles, knowing the user, speaking for the user in the development and sticking with the user focus. The following characteristics of OSS philosophy were found: the selection of the target community was based on one usability specialist scratching his own itch, the software was seen as a communal resource, collaboration was voluntary, interaction with the community was loose, usability specialists acted as bug reporters and designers gaining merit and reputation through contributing to the community.

This thesis contributes to the existing literature of introducing usability activities into open source software development by examining a new case study. Compared to the prior research, a new instantiation of OSS philosophy characteristic, “scratching your own itch” was found.

Keywords

usability, open source software, human-computer interaction, philosophy

Contents

1. Introduction

The purpose of the thesis is to examine how a usability project which introduced usability activities into an open source software (OSS) project affected the community's attitudes towards usability and usability activities, and to find out the co-existence of core HCI and OSS philosophies during the usability project.

Usability is one of quality aspects in information systems with sub-characteristics understandability, learnability, operability, attractiveness and usability compliance (ISO/IEC 9126-1). It plays a central role in technology acceptance models (e.g., Venkatesh & Davis, 2000) and in information success model (DeLone & McLean, 2003). Usability is a product of user interaction design, and it is recommended to evaluate user interfaces during user interaction design as early as possible in order to make the user interfaces more usable by involving participants (Sharp, Rogers & Preece, 2007). The methods to evaluate usability are, for example, usability testing (Dumas & Redish, 1999; Rubin, 2008; Krug, 2010) and inspection methods such as heuristic evaluation (Nielsen, 1994) and cognitive walkthrough (Wharton, Rieman, Lewis & Polson, 1994).

In Scandinavia, user participation has been used in system development for more than three decades. The reasons for participating users are: gaining knowledge of the systems, to develop realistic expectations, reduce change resistance and increasing workplace democracy as organization members have rights to participate in decision-making that concerns their work (Bjerknes & Bratteteig, 1995). User participation is also a characteristic in OSS development as some communities are democratic (Fogel, 2013). However, in meritocratic OSS communities, non-technical users do not have decision-making power (Bonaccorsi & Rossi, 2003) which is also a gatekeeping tactic (Rajanen, Lanamäki & Iivari, 2015).

It has been reported that open source software systems have poor usability (Nichols & Twidale, 2003; Hall, 2014) because achieving good usability is not the primary objective of OSS development (Çetin & Göktürk, 2008). Poor usability might occur due to cultural reasons (Hall, 2014) and usually usability has been neglected in OSS as developers are "scratching their own itch" (Raymond, 1999). However, people with less technical skills are using OSS more and more (Raze, Capretz & Ahmed, 2012), which continues to express the need for usability in OSS. With its techniques, the user-centred movement has been closing the gap between users and developers and OSS has used user interface ideas from commercial products, which both improve the usability of OSS products (Nichols & Twidale, 2003).

Institutionalization of usability refers to hiring usability specialists to a software project, conducting usability activities and sharing usability knowledge to developers. Usually institutionalization of usability occurs after a wake-up call which in the worst-case scenario can be that the product is perceived to be impractical or unusable. Institutionalization requires support from commitment and support from management. (Schaffer, 2004.)

A number of researches has been conducted about usability in open source software development. For example, Rajanen, Iivari and Anttila (2011) evaluated different approaches for introducing usability activities into open source software context.

Rajanen and Iivari (2013) examined the co-existence of core human computer interaction (HCI) and open source software (OSS) philosophies in two usability projects. Andreasen, Nielsen, Ormholt and Stage (2006) conducted an empirical study of the developers' opinions about usability and how usability engineering is done in some OSS projects. Twidale and Nichols (2005) examined how usability bugs are reported and resolved.

Still, more research is needed to understand the appropriate and culturally compatible approach to combine OSS and HCI philosophies (Rajanen & Iivari, 2013). In addition, both studies mention the other factors to be considered in the future research are different kinds of projects in size, domains, level of company involvement, governance styles and structures, to understand the possibilities of usability institutionalizing in OSS projects (Rajanen, Iivari & Anttila, 2011; Rajanen & Iivari, 2013).

This thesis contributes to the aforementioned research topics by contributing explanations of the impacts of a usability project to an open source software community, and how HCI and OSS core philosophies existed during the usability project.

The purpose of this thesis is to examine the impacts of a usability evaluation project called UKKOSS11 to the target open source software project, Concrete5, and the existence of human computer interaction and open source software philosophies during the project.

The research questions are:

1. RQ1: *What kind of impacts usability evaluations had on the community?*
2. RQ2: *What aspects of HCI philosophy existed during UKKOSS11 project?*
3. RQ3: *What aspects of OSS philosophy existed during UKKOSS11 project?*

The first research question attempts to answer how UKKOSS11 project affected the community. A questionnaire was done for one of the project leaders of community in the beginning of the usability project in 2013 and the project leader was asked the same questions in 2016, almost a year after the last reports were sent. Also the community website including forums and tutorials, IRC channel and GitHub issue tracker are included in the research material.

Questions and other observables to define the status of usability in terms of activities and discussion are that has there been previous usability activities in the OSS project, is there a To Do -list where HCI experts could contribute, are there already identified usability issues, do developers and community seem to know anything about the usability and usability activities, did usability activities result a "wake-up call" within the community and is there usability-related discussion within the project.

In order to answer to research questions two and three, UKKOSS11 project materials, and community's communication channels should be observed. The observed communication methods are IRC channel, community's website and GitHub issue tracker.

The research was conducted using case study research method (Yin, 2014) similarly to the studies of searching for a suitable approach to introduce usability activities into open source software development context (Rajanen, Iivari & Anttila, 2011) and HCI and OSS philosophies (Rajanen & Iivari, 2013). The research was conducted by following

the steps of the case study research process: plan, design, prepare, collect, analyze and share (Yin, 2014) including the material mentioned above.

This study contributes to the existing literature by adding another case study of introducing usability activities into open source software development. The results can be used further to analyze which kind of usability approaches are suitable for different types of open source software projects. The consultative approach used in the usability project resulted to some kind of a wake-up call in the community as it was seen as beneficial and the community asked for more usability activities, but no institutionalization of usability was found. In addition, the existence of the following core HCI and OSS philosophies was found. The found HCI philosophy characteristics were that usability specialists were user representatives in informative and consultative roles, knowing the user, speaking for the user in the development and sticking with the user focus. The existed characteristics of OSS philosophy were that the selection of the target community was based on one usability specialist scratching his own itch, the software was seen as a communal resource, collaboration was voluntary, interaction with the community was loose, usability specialists acted as bug reporters and designers gained merit and reputation through contributing to the community. Compared to the prior research about the existence of HCI and OSS philosophies in two cases, committer, developer and gamer roles, were not found, but in this study "scratching your own itch" characteristic existed.

The structure of this thesis is as follows. Next the prior research is discussed regarding to usability and open source software development. The usability topics are user interaction design, usability testing, usability inspection methods including heuristic evaluation and cognitive walkthrough, cost and benefits of usability activities, and lastly, core human computer interaction philosophies. The open source software development topics are community structures, motivation of developers, governance mechanisms, usability, usability bugs, and core open source software philosophies. The third section covers research material and methods. Findings of the study are presented in fourth section. The fifth section is the discussion and implications section of this study. Finally, the sixth section concludes this study.

2. Prior Research

This chapter discusses prior research of the main topics: usability and open source software development. First, the definition of usability is provided. Then, the focus is put on user interaction design and three usability evaluation methods: usability testing, heuristic evaluation and cognitive walkthrough. Then the costs and benefits of usability activities are discussed, and the last chapter about usability is human computer interaction philosophies. The second main topic is open source software development, and the sub-topics covered are community structures, motivation of OSS developers, governance mechanisms, usability in OSS, usability bugs in OSS and the core OSS philosophies.

2.1 Usability

Usability has a number of definitions. ISO/IEC 9126-1 (ISO/IEC 2500) standard defines usability as one of the main software quality characteristics. According to the standard, usability is further divided to five sub-characteristics: understandability, learnability, operability, attractiveness and usability compliance. The definition of the standard is "the capability of the software product to be understood, learned, used and attractive to the user when used under specified conditions". Another standardized definition is provided by ISO 9241-11 which defines usability as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use". Nielsen (1993) finds five quality components in usability: learnability, efficiency, memorability, errors, and satisfaction. Dumas and Redish (1999) state that usability "refers to how people work with the product" (p. 4) and that usability can be found in every product. They also describe usability as users' efficiency and ease of use while working on a task, which relies on four facts:

1. Usability means focusing on users.
2. People use products to be productive.
3. Users are busy people trying to accomplish tasks.
4. Users decide when a product is easy to use. (Dumas & Redish, 1999, p. 4).

Apart from ISO 9126-1 and ISO 9241-11, other ISO standards considering usability exits: ISO 13407 and ISO 18529. According to Marghescu (2009), these four standards have differences when looking at them from entity, stakeholders and phase in life-cycle aspects. From the viewpoint of entity, ISO 9126-1 focuses on software product, ISO 9241-11 focuses on software, hardware or service product in interactive systems, the focus of ISO 13407 on computer-based interactive systems, and ISO 18529 deals with life-cycle processes of computer-based interactive system, software and hardware. From the stakeholders' aspect, ISO 9126-1 is aimed at designers, developers, evaluators, maintainers and acquirers, ISO 9241-11 is targeted to designers, developers, evaluators and acquirers, ISO 13407 is aimed at project managers and all parties involved in human-centred system development, and ISO 18529 is meant for actors involved in design, use and assessment of life-cycle processes. The life-cycle phases the standard is meant to be used are: for ISO 9126-1, requirements, development, use, evaluation, support, maintenance, quality assurance, audits and acquisition, ISO 9241-11 considers design, development, evaluation and procurement phases, ISO 13407 covers the system

development life-cycle, and ISO 18529 focuses on design, development, uses and assessments of life-cycle processes of systems, software and hardware. Even though the standards have differences, they complement each other.

Usability is an important quality aspect in information systems success model and technology acceptance model. DeLone and McLean's (2003) updated information systems success model presents that user satisfaction, intention to use, and usage itself play an important role in gaining net benefits, which lead to increased intention to use and improved user satisfaction. First, information quality, system quality and service quality must be established and their quality assured in order to enable this positive cycle of benefits. In the technology acceptance model (TAM) by Davis (1989), perceived usefulness and ease of use improve the attitude toward using and behavioral intention of use, which consequently lead to actual system use. There are updates to TAM, TAM 2 (Venkatesh & Davis, 2000) and Unified Theory of Acceptance and Use of Technology (UTAUT) (Venkatesh, Morris, Davis & Davis, 2003), UTAUT2 (Venkatesh, Thong & Xu, 2012) and DTPB (Taylor & Todd 1995) which continue to express the need for usability.

Rubin (2008) lists five reasons for why some products and systems are difficult to use. The first reason is that the emphasis and focus during product development is put on the system instead of end-users. The second is that the target audience of technology has changed due to the penetration of technology to the mainstream consumer market, and development organizations have not reacted quickly enough to the change. The third reason is that it is difficult to design usable systems even though many organizations think that is sensible. The fourth problem is that while organizations have very talented teams and approaches to development, they fail to integrate them with each other. The fifth reason is that user interface design and implementation are different steps in the process, both requiring different skill sets. Nowadays the focus is on the design, but the engineers have more technically-oriented mind sets and skill sets (Rubin, 2008). Usability professionals have a number of, at least, 52 different titles, for instance: "HCI expert, usability engineer, interaction designer, user experience architect, cognitive scientist, etc." (Boivie, Gulliksen & Göransson, 2006, p. 601).

The commitment and support by management is an important factor for the success of introducing usability into software development. It is suggested to have a usability champion with understanding of usability and the person should provide coordination, resources and leading. The usability champion should evangelize usability within the organization and also have an eye for how to fit usability activities into the development process. The first step to apply usability-thinking into the development is usually a wake-up call which emphasizes the need for usability improvements. The worst type of wake-up call is that a software product shows to be impractical and unusable, and other types include executive or new staff insight, results of usability testing or expert reviews. After the initial activities, usability can be institutionalized even further by training personnel, although usability experts are recruited, as developers need generic-level knowledge about usability. The experts should be trained more extensively to know about usability strategies, methods, tools and infrastructure. (Schaffer, 2004.)

Next in this chapter, the following topics are discussed: user interaction design, which brings usability to products, then usability testing, two usability inspection methods called heuristic evaluation and cognitive walkthrough. Usability testing and both inspection methods were used in the UKKOSS11 project discussed later in the thesis. The last two topics are usability cost-benefit analyses and lastly, the core human computer interaction philosophies which summarizes the chapter.

2.1.1 User Interaction Design

According to Sharp and his associates (2007), user interaction design is crucial for designing products and services with good usability. They recommend user interface designers to first establish requirements by identifying needs, define usability and user experience goals and conceptual models. Also, they suggest to develop alternative versions, which suggests that the designs need version control. Interaction design process consists of four phases (see Figure 1):

1. Identifying needs and establishing requirements for the user experience
2. Developing alternative designs that meet those requirements
3. Building interactive versions of the designs so that they can be communicated and assessed.
4. Evaluating what is being built throughout the process and the user experience it offers. (p. 17)

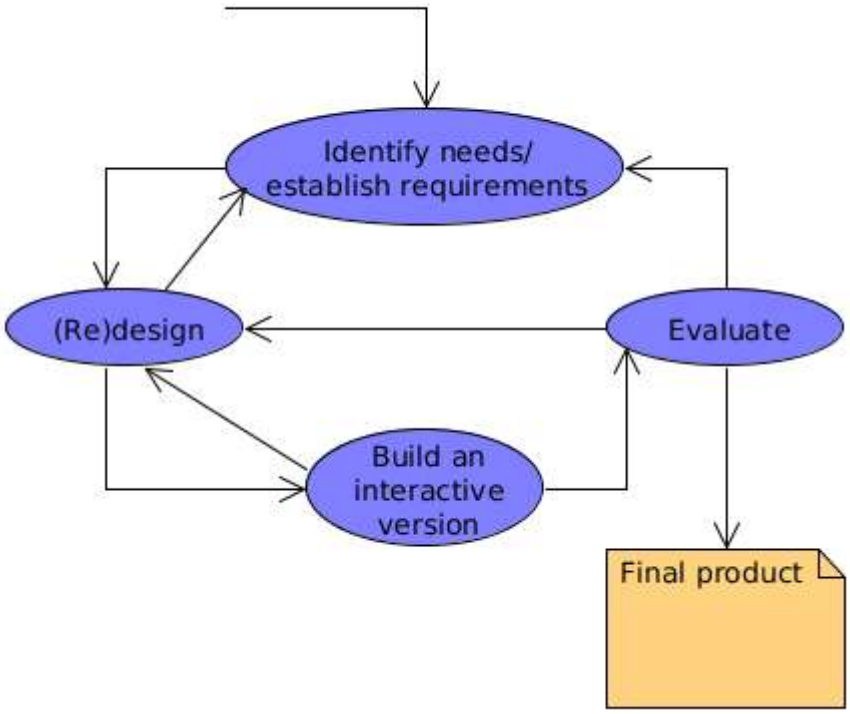

Figure 1. An interaction design lifecycle model. Adapted from Sharp et al. (2007).

First, analysis of current situation is expressed, after which the design and evaluate cycle occurs and the artefact moves to more rigorous level. The goal of the evaluation process is to identify usability-related issues which are used to repeat the process by establishing new requirements for next iteration of the process (Sharp et al., 2007). The methods to identify user interface problems are usability testing or inspections. After identifying problems, the development team redesigns the interfaces to fix as many problems as possible (Nielsen, 1994). If the evaluation shows green light, a final design alternative has been found successfully. There are four cases when to evaluate user interfaces: early design of artefact to clarify design ideas, evaluation of a prototype, refining or maintaining a product, and exploration of a new design concept (Sharp et al., 2007).

Next, the last step of the user interface design process, evaluation, is discussed in more depth through usability testing and two usability inspection methods: heuristic evaluation and cognitive walkthrough.

2.1.2 Usability Testing

The goal of usability testing is to gather data about how usable a software product is and how well the product corresponds to the original requirements set by the customer (Wahl, 2000) or "the business goals within the organization" (Rubin, 2008, p. 84). An example statement of a usability goal is that "*Users will be able to select the correct icon in less than 30 seconds with no more than one mistake*" and an example of a concern statement is "*Will users be able to select the correct icon quickly and accurately?*" (Dumas & Redish, 1999, p. 110). After defining the goals of usability testing, further planning becomes much easier as it defines the user group to be tested, test tasks, test data and equipment, and data to be gathered (Dumas & Redish, 1999). In addition to finding usability problems, usability testing includes to assess and prioritize found usability problems and develop recommendations to fix them (Rubin, 2008). Other benefits of usability testing are that it changes "people's attitudes about users" and "the design and development process" (Dumas & Redish, p. 32).

The complete usability test process consists of six major stages: developing a test plan, selecting and acquiring participants, preparing test materials, conducting the test, debriefing participants, and transforming data into findings and recommendations (Rubin, 2008). Next, each major stage will be discussed in more depth.

Developing the Test Plan

The first step of conducting a usability test is to develop a test plan. The structure of a test plan consists of the following sections: purpose, problem statement, user profile, method, task list, test environment and equipment, test monitor role, evaluation measures, and finally, report contents and presentation. (Rubin, 2008.)

The purpose section provides a high-level description of the need to perform the test at the given time. However, specific objectives or particular problems should not be included in the section. The aim is to support business goals and provide reasons from the organization's viewpoint, for example, to fix issues from customer support or a new policy which states that all products must be tested in the development phase. (Rubin, 2008.)

The problem statement section is the most essential individual section as it provides the issues and questions that should be resolved and describes the research and other activities related to the test (Rubin, 2008). Also, a starting point for this part is to consider "what aspects of the product might not be as usable as they should be" (Dumas & Redish, 1999, p. 105). Concern sources can be task analyses, quantitative usability goals, timely issues, heuristic analyses, expert reviews, or prior tests (Dumas & Redish, 1999). The section should address the problem statement as specifically as possible including measures (Rubin, 2008; Dumas & Redish, 2008). A possible consequence of an unclear problem statement is that the test fails to answer to the key concern of developers even though the test itself is conducted perfectly. Examples of problem statements such as "*Is the product usable?*" or "*Is the product ready for release or does it need more work?*" (Rubin, 2008, p. 85) are too unfocused and vague to be good statements because they lack measures and means to quantify results (Rubin, 2008; Dumas & Redish, 1999). Rubin (2008) provides five example problem statements for

software usability tests: “Are end users able to move freely between the two major modules?”, “Do novice end users inadvertently wander into the advanced features screen?”, “Is the response time a cause of user frustration or errors?”, “Is help easier to access via a hot key or via a mouse selection?”, and “Do the screens reflect the end user's conceptual model?” (p. 86). The origins of problem statements come from a discussion with the developers and other participants in the development process (Rubin, 2008).

User profile section provides a description of the end user(s) of the product (Rubin, 2008). The participants should represent the actual end-users of the product (Dumas & Redish, 1999). User profile consists of characteristics such as general computer experience, education level, age, gender, learning style preference, education major, operating system experience, and computer interaction experience, to give a few examples. Furthermore, each characteristic should have a range of values and frequency distribution, both considering the user population (Rubin, 2008).

The goal of the method, in other words, test design, section is to provide a description of how the test is executed with the participants. It is a summary of the test design meaning that an overview of each step of the test should be described. The description should be so accurate that an observer should clearly know what is going to happen during the tests. Moreover, the section should be written and visualized in such detail that others can understand, comment, and make suggestions. Moreover, this section of the test plan helps a test developer to focus on the prior-test activities and to acquire the needed material. The benefit of a well-written method section allows others to repeat the tests in a similar manner. The factors that should be considered prior to design are resources, creativity and the following constraints: time, money, management and support of development team, easiness of acquiring participants. (Rubin, 2008.)

There are multiple test designs that consider in which order participants test product modules. The designs are called: independent group design, within-subjects design, randomized order of module presentation, testing multiple product versions and testing multiple user groups. Independent group design refers to a test design in which a unique set of participants is used for each software module. This method mitigates learning effects between similar tasks, which consequently might help users to perform better in later tasks. In addition, if test tasks are really long, this method should be used to avoid increased fatigue of test participants. However, this method requires a relatively huge number of participants, for example, a test case consisting of three modules including four test participants per module requires to acquire 12 participants in total. (Rubin, 2008.)

The within-subjects design means that all participants go through all the modules in the same order. Thus, tests can be executed with a small number of test participants, but on the other hand, there is a problem in considering the transfer of learning effects, which can be mitigated by counterbalancing meaning that the order of test tasks are randomized or balanced, referring to another separate test design called randomized order of module presentation. As mentioned earlier, the method is used to mitigate transfer effect. However, if the task order is sequential in a real use case, another problem arises as the logical order breaks. Therefore, it is suggested to test the designed task order, but still the transfer effect should be addressed. (Rubin, 2008.)

In addition to testing different tasks, it is possible to test multiple product versions, for instance, to see which version is a better candidate for further development or release. This happens by using four participants per targeted user group in the test. For example, if a product has two user groups, supervisors and technicians, a performance of each

user group per product version can be tested. As in the within-subjects design method, this method also involves the problem of transfer of learning effects when using same participants in both versions, meaning that participants might learn to perform differently in the other version's tests resulting in bias. However, counterbalancing can remove the bias when having an even number of test participants. The last test design method is testing multiple user groups to see whether there are differences in user groups or even inside the user groups as one group might consist of novice and experienced users. A noteworthy issue is that a user population should include at least four participants, and thus testing different variables of user characteristics requires more participants. (Rubin, 2008.)

Rubin (2008) presents eight ground rules for achieving an adequate level of experimental rigor: employ an adequate number of participants, consistency, confirm the characteristics of your participants, note any unusual problems with the test, have specific goals or objectives in mind, conduct a pilot test, keep it simple, and make the testing environment as realistic as possible. Next, all of these rules are explained in more depth. Based on the purpose of the usability test, it is important to acquire a right number of participants. When conducting a less formal test, using four to five participants reveal 80 percent of major usability-related issues on average (Rubin, 2008). Steve Krug (2010) suggests do-it-yourself practitioners use three participants because they still are likely to find the most critical usability issues, they are easier to recruit, and the tests do not take longer than a day. Donahue (2001) suggests to use 3-5 participants to test the usability of prototypes. However, if the tests must achieve a certain level of statistical validity, at least 10 participants per condition should be used to avoid too much variance in the results (Spyridakis, 1992).

When it comes to experimental rigor, consistency is an essential matter. The test sessions should be executed as identically as possible by having the same material and conditions. If scripts are used, the moderator should read every word of them without omitting or adding anything. Also, checklists prevent moderators from forgetting activities or instruction during sessions. It is also recommended that the same person conducts all the sessions. Before starting the test sessions, it is important to ensure that the characteristics of participants fit the description of the user group. This can be done by filling out background questionnaires, after which a decision is made whether to send the participant home or start the session. If something goes wrong during a test session, only minimal changes should be made if testing is not part of fast iteration exploratory tests. If the product has to be changed during a test session, the participant should not see any information regarding the product, and thus, he or she should be sent to a waiting area. Usability tests should always have a specific goal or objective. Even an exploratory test without goals or objectives is unfocused and informal. It is recommended to search for particular problems. Otherwise, there is temptation for a usability tester to find what he or she would like to find. A very important step before the actual test sessions is a pilot test. The purpose of a pilot test is to reveal problems in the product, test method and test design. If problems are found, there is a possibility to fix them before the main tests. It is recommended to keep the tests simple, particularly if the people conducting them are new to usability testing. By making simpler tests, it is easier to keep them consistent. Furthermore, usability test should be conducted in the early phases of development and often. The testing environment should be as realistic as possible, resembling the actual working environment. A busy office environment can be mimicked by simulating the noise caused by ringing phones and other kinds of interruptions. If the participant relies on the aid of a help desk, one person from the test group should simulate that. (Rubin, 2008.)

The task list section consists of the tasks that are included in the usability tests and will be performed by the participants (Rubin, 2008; Dumas & Redish, 1999). Developing the tasks list includes two separate steps: the first step happens in the early phase of the development when the list is available only for the development team for development, documentation and review, and later, the list is expanded into a more elaborated version including task scenarios providing the realistic details and context. Finally, the task list should include the following: a brief description of the task, the materials and machine states required to perform the task, a description of successful completion of the task including the criteria for a successful performance, and benchmark timings that establish the maximum time limits for performing (Rubin, 2008).

An example task scenario consists of four task components with their descriptions: task, machine state, successful completion criteria, and benchmark. Example descriptions for the components are: for task: "load paper into the copier", machine state: "copier with four labels attached and an empty cassette tray", successful completion criteria: "test subject loads paper into cassette after first fanning the stack of paper", and benchmark: "loads correctly within one minute" (Rubin, 2008, p. 101).

The test environment and equipment section describes where the test sessions will be executed and what kind of equipment is involved in the process. The ground rule is that the better the simulation, the more precise prediction for the product's performance in the workplace. An example of a test environment is a busy and noisy office with ringing phones. The equipment used by the participants in such settings could be phones, computers and printers, etc., which aid to complete tasks required by some office tasks. (Rubin, 2008.)

The test monitor role is an optional section. It helps to make clear what the test monitor does, especially when there are observers with no prior experience on usability test sessions. Moreover, it specifies cases in which the test monitor is allowed to ask questions and when not to intercept participant's work. (Rubin, 2008.)

The evaluation measures section defines which kind of data will be collected. Mainly, there are two types of measures: performance and preference data. The former represents measures of behavior and the latter represents measures of participant's opinion or thought process and it can include rankings and answers to a questionnaire. Some example measures of performance data are: time to complete each task, number and percentage of tasks completed correctly with and without assistance, count of incorrect selections, and count of user manual accesses. Example measures of preference data include usefulness of the product, how well does it match the expectations, ease of use or learning, or comparisons between two variables, for instance, preferences and rationale for a product or version versus another product or version. Also quotes such as "I loved it!" belong to the preference data (Dumas & Redish, 1999; Rubin, 2008).

The report contents and presentation section sum up the main sections and describe how to communicate test results to the development team. The section consists of the test report structure and a description of how the results will be presented. For example, there might be an informal meeting after the tests, presenting critical findings, and after the analysis and reporting phases are over, an official meeting can be arranged with all the stakeholders. (Rubin, 2008.)

Selecting and Acquiring Participants

The second stage of conducting a usability test is to select and acquire participants. It is an essential part of the process as it brings validity to the whole process, and thus increases the value of the process. The step begins by identifying and describing relevant skills and knowledge of the targeted user group. The outcome is called a user profile or user characterization, which preferably should have been done in the early phases of the product's development process (Dumas & Redish, 1999; Rubin, 2008). Next, the major activities of this stage of the process are explained.

First, the user profile of the target population is documented by describing their most essential skills, knowledge, demographic information, and other factors. A generic user characterization consists of characteristics such as personal history (age, gender, attitude toward computers or your type of product, left or right handed, learning style, and attitude toward high technology), education history (highest grade completed, subjects studied and major), computer experience (total time using, frequency of use, types of computers used, operating systems used, and types of screen interaction used), product experience (total time used, frequency of use, types of tasks performed and frequency, and types or brands used) and occupation history (current and past job titles, responsibilities, training classes taken, and time with the current company). Scale and distribution should be presented for each characteristic (Rubin, 2008). According to Dumas and Redish (1999), characteristics that all user share should be developed first, and then those characteristics that differ in the user population. Information for a user profile can be found from several sources: marketing department (Dumas & Redish, 1999; Rubin, 2008; Donahue, 2001), product designers (Dumas & Redish, 1999) or managers in R&D, the functional specification or product requirements, structured analyses or marketing studies, and competitive benchmarking and analysis group (Rubin, 2008). According to Rubin (2008), if these sources cannot provide information, there is also a chance to ask or verify user-related information directly from the end users by making phone calls, visiting user sites or by conducting a survey.

When making the user profile, a clear distinction between buyer and end user must be made because sometimes information about the buyer is used in the user profile (Rubin, 2008; Donahue, 2001). Another case causing problems is that the end user of the product and the end user of the documentation can be different persons as an organization might have system administrators who receive the documentation instead of the end users (Rubin, 2008).

The user profile gathers all the information about the whole population, but often it may be beneficial to group users based on their characteristics, usually by occupations or job titles, to make the product more usable for everyone, even though it is used differently by each group (Dumas & Redish, 1999; Rubin, 2008). For instance, managers and other personnel should be able to use the same product even though the groups have different backgrounds (Rubin, 2008). Therefore, there is a need to organize usability tests for each group (Dumas & Redish, 1999; Rubin, 2008). If different groups are tested, the number of participants from each group should be in balance (Rubin, 2008). When defining different user groups, it is very important to avoid so-called buzzwords such as novice and expert as they mean different things to different people. Thus, more specific descriptions, preferably measurables, can be used. For instance, a novice can be defined as a person who has used the system less than six months (Dumas & Redish, 1999; Rubin, 2008).

The number of participants that should be acquired depends on five factors: the level of confidence of results, number of available resources, availability of the type of

participants, duration of test sessions, and time required to prepare for the test. In general, time and resources are the major restrictions when conducting usability tests. If the plan is to test the product multiple times during the development life cycle, it is possible to test with less participants (Rubin, 2008). Dumas and Redish (1999) suggest to use three to five participants per subgroup.

It is suggested to include a few less competent users in usability testing even when they do not fit perfectly the description of the targeted user group, but still could potentially use the product. By involving less competent users, it is possible to reveal problems considering intuitiveness, orientation, documentation, and organization through redesign (Rubin, 2008). Dumas and Redish (1999) suggest to acquire participants from both ends of the range rather than from the middle in order to obtain more accurate information.

Also internal participants, i.e., people who work for the company that produces the product, can be used in usability tests even though they do not represent the primary participants (Rubin, 2008). If the product is for internal use of the company, then internal participants are the actual user group (Dumas & Redish, 1999). They may fit the user profile except for the fact that they do not work in the right company, which causes the only problems when involving internal participants in the tests as they have different corporate culture, may possess exclusive information, and they expect to have a successful product. Thus, they are not the best persons to take part into actual tests requiring high confidence level, but they can be used to try out the test, to conduct early exploratory research, and to conduct best case testing. However, there is a pitfall in testing the best people as they can work around the areas that might cause critical problems to other users (Rubin, 2008).

After a user profile is defined and an adequate participant selection list including their preferred characteristics is done, it is time to acquire the participants (Rubin, 2008). There are multiple sources from which they can be acquired: employment agencies (Rubin, 2008), other agencies, advertisements, networking, professional associations (Dumas & Redish, 1999), market research firms, existing customers from in-house lists or through sales representatives, personnel departments, college campuses, user groups and qualified friends (Rubin, 2008).

The participants should always be provided with something to compensate their time spent on usability tests by paying them or providing other kind of incentive (Dumas & Redish, 1999; Rubin, 2008). It can be the company's merchandise, such as pens or t-shirts, but when it comes to testing with professionals, they should be compensated according to their hourly rate. A special case occurs when you are using your company's clients. Then the compensation can be settled according to the relationship between the companies (Rubin, 2008).

Preparing the Test Materials and the Test Team

Preparing the test materials is the third and one the most effort-requiring stages of usability tests. The purpose of the stage is to prepare for communication with participants, data collection, and to comply with legal requirements (Rubin, 2008). During this stage, the following, most common material should be prepared: screening questionnaire, orientation script, background questionnaire, data collection instruments, nondisclosure agreement and tape consent form, pretest questionnaire, task scenarios, prerequisite training materials, post-test questionnaire (Dumas & Redish, 1999; Rubin, 2008) and debriefing topics guide (Rubin, 2008). Also the product including hardware, software and sample data should be prepared before the tests (Dumas & Redish, 1999).

However, the materials vary from test to test, and thus some material can be left out. It is recommended to develop the material as early as possible because it helps to organize the tests (Rubin, 2008). Dumas and Redish (1999) note that the materials should be prepared for a pilot test, but should be copied after it. Next, the definition and purpose of each material is explained.

The purpose of the screening questionnaire is to ensure that the participants qualify for the test and that they can be selected. User profile and test plan provide the background information for the questionnaire. Usually, the questionnaire is done over phone, but it can also be done via mail or in person. (Rubin, 2008.)

The orientation script, also known as the test script, is an artefact used to communicate with participants. The purpose of the script is to describe what happens during a session and eases the minds of participants by expressing that the object being tested is the product, not the participant. Thus, the script is important especially when a participant does not have prior experience in usability tests. The preferred timing to read the script is at the beginning of a test session, but the script can be also placed in the waiting area. (Rubin, 2008.)

The background questionnaire is used to understand the behavior and performance of a participant by obtaining information of his or her prior experience before a test session. The background questionnaire is similar to a screening questionnaire but more detailed. The purpose is to reveal participants' experiences, attitudes and preferences that might have an effect on their behavior or performance because it is helpful to know whether participants have used similar products earlier and possibly for how long. Overall, the questionnaire serves two purposes: to ensure that participants fit the description and to provide a summary of a participant for the test team (Dumas & Redish, 1999; Rubin, 2008).

Data collection instruments, also known as data loggers, are an aid to collect data related to the test objectives provided in the test plan (Dumas & Redish, 1999; Rubin, 2008). Data collection tools should be simple, reliable, and using them should not interrupt test sessions. The collected data is divided into two categories: performance and preference data. Performance data is a measurable objective and related to the participant's behavior. Examples of performance data are: time to complete a task, number of errors, percentage of tasks completed successfully. Preference data is subjective and measures participant's feelings or opinions. Examples of preference data are: preference of version A versus version B, suggestions to improve the product, and ratings or rankings of the product (Rubin, 2008).

The purpose of the nondisclosure agreement and the tape consent form is to protect both the company and the participant (Dumas & Redish, 1999; Rubin, 2008). Nondisclosure agreements are typically used when people outside a commercial software company may encounter sensitive product information. Tape consent forms are used to protect the confidentiality of participants. Moreover, usage of showing participants' full names should also be avoided, and the recorded tapes should not be shown to other people except the stakeholders. Both agreements should be done before a test session begins, and participants should be informed that their session will be recorded (Rubin, 2008).

The pretest questionnaire addresses particular test objectives and it is given before a test session. Mainly, it serves four purposes: first, to discover the participant's attitude and initial impressions about a product, secondly, to discover the participant's opinions about the usefulness of a product, thirdly, to categorize the participant to a particular test group, and finally, to enquire participant's prior knowledge before using a product. (Rubin, 2008.)

Task scenarios list what participants have to do during a test session. It is a more detailed version of a task list defined in the test plan. Task scenarios should include the goal(s) of activities, moves, actual data and names, system's state before starting a task, and outputs from the system during a task (Rubin, 2008). Scenarios should be written in user's, not the product's, words and so that every participant understands it (Dumas & Redish, 1999).

Prerequisite training materials are optional means for test sessions. The purposes for training participants prior to test sessions are that their skill levels can be established to a minimum level and their learning curve can be measured. The latter can be used when attempting to assess the ease of use of a product. (Rubin, 2008.)

The purpose of post-test questionnaires is to collect preference type information from test participants to understand the strengths and weaknesses of a product. Because the type of gathered information is preferences, a post-test questionnaire includes questions related to opinions and feelings about ease of use and learning. The method is supposed to be used for the complete population of test participants by asking all participants the same questions (Dumas & Redish, 1999; Rubin, 2008). Dumas and Redish (1999) suggest that the debriefer should leave the room when the participant fills out the questionnaire because the participant might pay more attention to the questionnaire by doing so.

The debriefing topic guide provides a structure for the debriefing sessions with a participant at the end of a test session after completing the actual tests with a product. Unlike the post-test questionnaire, which includes very specific questions, the debriefing topics guide helps to structure general topics for a debrief session. Also, notes made during a test session can be used alongside the premade debriefing topics guide during a debriefing session. (Rubin, 2008.)

Furthermore, Dumas and Redish (1999) recommend to make plans for disasters. An example of a disaster during a test session is that the software crashes, and because of that, the working file corrupts, thus, a backup file should be immediately available.

Dumas and Redish (1999) recommend that the team which conducts the usability tests understand their roles and practice them before starting the actual tests. The test team consists of two to five people depending on their skill level, laboratory settings and participant's movement. Two people are enough if they are skilled enough and the participant does not have to move around during the test. Sometimes client's technical staff participate in the tests increasing the number of people in the laboratory. The roles of the test team include: test administrator, briefer, camera operator, data recorder, help desk operator, product expert and narrator.

Conducting the Test

Conducting the test is the fourth stage of a usability test process. The focus of this section is put on the classic situation of having one participant per test session. There are four types of tests which can be conducted using one participant at a time: exploratory, assessment, validation and comparison. It is emphasized that an integrated product should be tested, even though it is possible to test each component separately. Components can be, for example, documentation, company's hotline or a software product. (Rubin, 2008.)

Before explaining checklists and detailed descriptions of activities, ten guidelines considering test sessions should be noted: "monitor the session impartially", "be aware

of the effects of your and body language", "treat each new participant as an individual", "don't rescue participants when they struggle", "if you make a mistake, carry on", "make sure the participants are really finished with a task before going on to the next one", "use humor to keep the session relaxed, and indicate to the participants that there is no right or wrong response", "if appropriate, use the thinking aloud technique", "probe and interact with the participant as appropriate", and "assist the participants only as a last resort" (Rubin, 2008 pp. 215-222).

Next, an explanation of each guideline is provided. A usability tester should present the product neutrally to participants, and show that he or she is not interested in the results. Also, when participants make action, a tester should speak and behave neutrally, whether the action is correct or incorrect. A test moderator should be aware of the effects of his or hers voice and body language. During test sessions, a usability tester might have unintentional influence on a participant through voice and body language. For example, a raised voice pitch might refer to agreement and body language can have an effect on participants, as moving closer is a signal of acceptance and on the contrary, moving further away from the participant means rejection. Each participant should be treated individually as they are unique and an opportunity for the product. Therefore, it is recommended to take a break between sessions and forming an assembly line should be avoided. Participants should not be rescued when they face problems in test tasks. Particularly test monitors with no experience have tendency to help participants due to the natural empathy of humans. Instead of helping participants, they should be encouraged to say why they are struggling. The so-called 'thinking aloud' technique is used to encourage participants to express themselves verbally during tests. A worst case scenario of rescuing a participant is that there might be a critical usability issue in the product, which could cause calls to the company's hotline in the future. Moreover, long struggles by participants are strong evidence for a developer that there are flaws in his or her product. If a test monitor makes a mistake by revealing information in the wrong place, the monitor should not panic, but carry on as nothing happened. If a mistake occurs, it invalidates only a small part of the test. Sometimes the participant does not even notice a mistake. Sometimes participants are not sure when they have completed a task and might move to the next task. Furthermore, a test monitor might make notes at the end of a task and a participant thinks he or she is done. An aid to remove this problem is to tell participants to signal when they are finished with a task. It is recommended to keep sessions relaxed by using humor and to let participants know that there is no right or wrong response. As there are multiple people watching the participant, it is important to use humor to get them relaxed. Moreover, if the participants find something funny in the product, a monitor should laugh with him or her instead of being defensive. Thinking aloud technique is recommended to be used when appropriate. The technique can be applied by asking the participants to express verbally what they are thinking while performing a task. This enables to capture their thinking process and to reveal their feelings and conceptual models. The advantages of thinking aloud technique are that it is easier to capture preference and performance information, participants might focus and concentrate better, and it is easier to find the root causes for incorrect behavior. However, the technique has also negative effects. Some people do not find the technique natural and feel that it distracts their performance. It also slows down participants' thinking process. In addition, it might be too exhausting to think aloud in long test sessions. (Rubin, 2008).

The amount of probing and interaction with participants should correspond to the type of test based on the test goals (Dumas & Redish, 1999; Rubin 2008). When conducting an experimental or validation test, the amount of interaction should be minimized. On the contrary, in other types of tests, the level of interaction with participants should be high in order to find out the causes of preferences and performance (Rubin, 2008). For

example, when testing a prototype, a designer should get as much information for diagnosis as possible, and thus there should be a lot of interaction (Dumas & Redish, 1999).

Assisting participants to complete a task should be the last solution to rely on because assisting affects directly to the test results. When analyzing the results, for instance, independent completion rates, the cases in which a participant has been assisted should be distinguished from the results in which no assistance was provided. Nonetheless, there are situations in which assistance can be given: when a participant is completely lost or confused, when a participants feels uncomfortable, very frustrated or may give up, when the product is not finished, and when a serious bug occurs requiring repair. (Rubin, 2008.)

Rubin (2008) suggests to make three different checklists after preparation work is done and to use them in different periods of time before test sessions. The first checklist should be used approximately two weeks before the first test and it includes four steps: "take the test yourself", "conduct a pilot test", "revise the document" and "check out all the equipment and the testing environment" (p. 225). The second list should be used one day prior to the test, including five activities: "check that the video equipment is set up and ready", "check that the product, if software or hardware, is working", "assemble all written test materials", "check on the status of your participants" and "double-check the test environment and equipment" (p. 226). Lastly, the third list is used during the test day including a number of activities: "scan your customized checklist", "prepare yourself mentally", "greet the participant", "have the participant fill out and sign any preliminary documents", "read the orientation script and set the stage", "have the participant fill out any pretest questionnaires", "move to the testing area", "establish protocol for observers in the room", "provide any prerequisite training if your test plan includes it", "either distribute or read the written task scenario(s) to the participant", "record the start time, observe participant, and collect all critical data", "have the participant complete all post-test questionnaires", "debrief the participant", "thank the participant, provide any remuneration, and show the participant out", "organize data collection and observation sheets", "using a tape recorder, summarize your main thoughts about the test results", "provide adequate time between test sessions" and "prepare for the next participant" (p. 227-228).

Debriefing the Participant

Debriefing the participant is the fifth stage of conducting a usability test. However, it can be argued whether the stage is an extension or a stage, depending on whether debriefing sessions are executed iteratively after each test task or incrementally after the session is executed. The purpose of the stage is to understand why errors occur by allowing participants to explain what a test monitor does not see and how to fix usability-related problems found during the test sessions. Debriefing sessions can be held at test premises or in a separate area. Some participants may be shy, and it is recommended to organize debriefing with them outside the test room, but still, observers should be allowed to listen to the debriefing session. (Rubin, 2008.)

Rubin (2008) presents two guidelines for interrogation: "[n]ever make participants feel at all defensive about their actions or their opinions" and "while questioning a participant, do not react to participant's answers one way or another" (p. 244). The first guideline means that neutral discussion should be established without a test monitor being prosecutive or defensive about the participant's performance. The latter guideline proposes that the test monitor should not make proposing reactions to participant's answers indicating that one answer is better than another (Rubin, 2008).

According to Rubin (2008), there are nine basic debriefing guidelines. The first guideline is to think about the session and questions related to it while the participant answers the post-test questionnaire. The second guideline is to review the post-test questionnaire meaning that the participant gets a little break while the test monitor reviews the questionnaire, which can yield to further questions using unexpected answers. It is recommended to make notes of the interesting answers. The third guideline is to start debriefing by letting the participant say freely what is on his or her mind. This lets the participants say freely what has worked or if the participant is frustrated, he or she has a chance to express it. The fourth guideline is to start questioning about high-level issues before moving to specific issues, which is the fifth guideline. The sixth guideline is to ask questions based on the review and notes of the post-test questionnaire discussed in the second guideline. The seventh guideline is to put the focus on "understanding problems and difficulties, not on problem solving" (p. 247). An end user knows whether a product meets his or her needs, but solving a problem requires a different kind of skill set. The eighth guideline is to let observers discuss with the participant. Lastly, the ninth guideline is to settle whether the participant can be contacted if further questions arise (Rubin, 2008).

Transforming Data into Findings and Recommendations

The sixth and last stage of conducting a usability test is transforming data into findings and developing recommendations. The stage is divided into two separate processes: the first process is a preliminary analysis and the second process is a comprehensive analysis. The purpose of the first process, preliminary analysis, is to find the most critical usability-related issues and communicate them to the developers in order to fix the issues as soon as possible. This way, the issues can be fixed without having to wait for the final report. The second process, a comprehensive analysis, requires more effort than the first process. The final artefact of the process is a report including the preliminary findings with other findings found in the analysis phase. Already delivered preliminary findings can be updated in the final report. If a usability test group is not sure whether a finding is a problem or not, it should be marked accordingly to the report (Rubin, 2008). Comprehensive analysis includes four steps: "compile and summarize data", "analyze data", "develop recommendations" and "produce the final report" (Rubin, 2008, p. 258). Next, the steps are explained in more depth.

The first step of comprehensive analysis is to compile and summarize data. The step includes seven sub-steps that should be done in the following order, but in practice, there might be some flaws in the workflow: begin compiling data as you test, "create summaries", "summarize performance data", "task accuracy", "summarize preference data", "compile and summarize other measures" and "summarize scores by group or version" (Rubin, 2008, pp. 259-273). The first sub-step starts already in the testing phase as it is recommended to compile data after each session in order to visualize that correct data is collected with appropriate accuracy as stated in the test objectives. Compiling data after tests speeds up the overall process. The second sub-step, create summaries, begins after all test sessions are executed. At this point, it is suggested to create summaries, transfer them to a summaries sheet and provide descriptions as the purpose is to see what has happened during the tests at one glance. Also, if the purpose of the test is to compare between two variables, for instance, groups or product versions, summaries provide the differences. The third sub-step is to summarize performance, i.e., task timings. The most important descriptive statistics for task timings are mean time to complete a task, median time to complete a task, range of completion times and standard deviation of completion times. The fourth sub-step is task accuracy and many statistics can be used to measure it, for example, number of errors, errors of omission,

errors of commission, percentage of participants performing successfully within the time benchmark, percentage of participants performing successfully or percentage of participants performing successfully with assistance. The fifth sub-step is to summarize preference data which originate from surveys, post-test surveys and debriefing sessions. For limited choice questions, it is recommended to sum the answers. The answer for free-form questions and comments should be categorized to positive and negative responses. Lastly, the interviews of debriefing sessions should be transcribed, especially the critical comments. The sixth sub-step is to compile and summarize other measures such as number of times help was accessed, number of times the manual's index was accessed and points of hesitation in the manual. Finally, the seventh sub-step is to summarize scores by group or version if the test design requires to test more than one group or product version (Rubin, 2008).

Analyzing the gathered data is the second step in transforming data into findings and recommendations. The input for this process is the summaries created in the prior step. The purpose of this step is to analyze the summaries in order to identify problems in the product and later, to create recommendations to fix the found problems. Overall, the summaries provide clues in which tasks the problems occur, and then a usability expert moves to the more specific details of a task to find the sources of the problems (Rubin, 2008). More specifically, the data analysis step includes six smaller steps: "identify and focus on those tasks that did not meet criterion", "identify user errors and difficulties", "conduct a source of error analysis", "prioritize problems by criticality", "analyze differences between groups or product versions", and use inferential statistics (Rubin, 2008, pp. 274-280).

The first sub-step of transforming data into findings and recommendations is to identify and focus on the tasks that did not meet the benchmarks, if they are set (Rubin, 2008). If no benchmarks have been set, a threshold of 70 percent success criterion for a task, which "represents a reasonable balance between being too demanding and too lax" (Rubin, 2008, p .274). The tasks that did not meet the criterion should be marked difficult or problematic. After identifying difficult tasks, it is time to put focus on them for further analysis (Rubin, 2008).

Identifying errors made by users and difficulties is the second sub-step of creating findings and recommendations. At this point, it is recommended to look for the causes that result to wrong performance. An example of a user error is that a participant enters a value in the wrong field. (Rubin, 2008.)

The third sub-step, which conducts a source of error analysis, requires most of the effort of the post-test activities. The purpose is to find the source of error by identifying a component, multiple components or other causes that resulted in incorrect performance. A wrong interpretation of a problem source results in an inaccurate recommendation. Thus, it is important to be careful and if necessary, investigate all available material and information such as tapes, notes, yours and observers' memories of test sessions, understanding of product's functionality, user-centered design, participants' background and discussion with other observers. Also, it is recommended not to create recommendations before each source of error has been identified (Rubin, 2008). Sources of errors can be found in the summary sheets or by examining outliers (Dumas & Redish, 1999).

After creating a list of error sources, it is time to move on to the next sub-step: prioritize problems by criticality. The equation to assess criticality is: criticality equals severity plus probability of occurrence. The purpose of the prioritizing activity is to structure the development team future so that most critical issues are fixed first. The severity ratings

use a four-point scale from one to four, one being the least severe rating and four being the most severe rating (Rubin, 2008). When it comes to severity ratings, Dumas and Redish (1999) use term level. More specifically, the severity descriptions from one to four are: irritant, moderate, severe and unusable. Also the frequency rankings have four values. Value one represents estimated frequency of occurrence of less than 10 percent of the time, value two corresponds to a range from 11 percent to 50 percent, ranking three refers to a range of 51 percent to 89 percent, and lastly, four is the highest ranking with estimated frequency of occurrence of 90 percent or more. It is important to note that it is difficult to reach the most accurate value, and thus, it is enough to make your best guess. An example of calculating the criticality of an issue is that if its severity is moderate, referring to a severity ranking of 2, and its estimated frequency of occurrence is 35 percent, referring to frequency ranking 2, the criticality would be 4 (Rubin, 2008). Dumas and Reddish (1999) use scope of error as an aid to assess the severity: global issues are more severe than local issues.

If the type of usability test is comparative, an analysis of differences between groups or product versions sub-step should be executed. The most important activity at this step is to recognize the sources of errors, in particular the types of errors, between groups or product versions. Sometimes it is very challenging to choose the winner of product versions if the differences are not clear. (Rubin, 2008.)

Sometimes more advanced techniques, such as inferential statistics, can be used to make better analysis. However, statistical techniques depend on various factors, for example, it is important to manipulate one factor, and thus, if statistical techniques are used, it is important to address the usage already in the design phase of the usability test. Another challenge is to use a minimum number of test participants in order to remove variance bias. It is recommended to use at least 10-12 participants for a reliable statistical result. (Rubin, 2008.)

At this point the analysis phase is completed and it is time to move on to developed recommendations which is the third step of transforming data into findings and recommendations phase. The input for this activity is all the gathered and analyzed information. Developing recommendations is an interpretative activity as each person interprets available information differently, and consequently, the developed recommendations may differ from each other. Thus, it is beneficial to discuss the problems and possible solutions with designers. (Dumas & Redish, 1999; Rubin, 2008.)

Rubin (2008), Dumas and Redish (1999) recommend to put the focus on solutions that have the most effect on the product. For example, changing a field or a short paragraph of text may not affect to the usability of the product as much as changes to navigation or other global changes.

When developing the initial version of recommendations, it is recommended to not pay attention to political issues, meaning those issues that will not have support from the management, even though later in the test process those considerations are reasonable. First, it is recommended to stay as objective as possible and to think about the users of the product, which results to the best solution recommendations. Occasionally, it is impossible to implement all the recommendations for the next release of the product due to time constraints. Therefore, short and long-term recommendations should be provided, both having their own sections in the report. (Rubin, 2008.)

A single usability test cannot cover the product completely and like research in general, many new questions arise or are left unanswered. Also, other techniques could be used in the same user tasks. Thus, it is proposed to write a section in the report advising which questions should be answered or which techniques should be used in further

usability tests. Even though it is recommended to prioritize and cover the most critical issues first, all identified issues should be covered. It should be addressed that the document will be read and referred by others in the future. (Rubin, 2008.)

The fourth step is to produce the final report. It is recommended to deliver a summary of the most critical issues to the development team before producing the final report (Rubin, 2008). Some stakeholders might already know the critical issues as they might have participated in the usability tests (Dumas & Redish, 1999). Still, at least a brief report should be made even though the development team is aware of each usability issue (Rubin, 2008). Other than a written report, other means to communicate results exist: communicating verbally or visually (Dumas & Redish, 1999).

The purpose of the test report is to "support and initiate change, direct action, provide a historical record, and educate" (Rubin, 2008, p. 289). Another noteworthy issue is that it will be read by others (Dumas & Redish, 1999; Rubin, 2008), and thus, it should not be written in a manner that only the person who wrote it understands it. Overall, the report begins by stating the purpose of the test and the preparations, the middle part of the report addresses what occurred during the tests, and lastly, the implications and recommendations (Rubin, 2008).

Rubin (2008) suggests to include five sections to the report: executive summary report, method section, results section, findings and recommendations, and appendices. The executive summary section should take a page in maximum consisting of a summary of test logistics, the most essential findings with recommendations to fix them, suggestions for future research, and the benefits of the test. The method section is a description of the type of the test, how the test was done, user profiles, and what kind of data was collected and how. The section may be very similar to the test plan if no changes occur. The results section presents summaries of quantitative and qualitative results. It is recommended not to include the raw data, but rather provide the results in as readable form as possible. The findings and recommendations section is a discussion of how the findings should be fixed. The findings and recommendations should be clearly visible for the reader, therefore it is suggested to present them in lists or tables. The appendices section should be used to include all raw data collected during the tests such as questionnaires and scripts.

The report should be revised (Dumas & Redish, 1999) and thus, the last guideline for reporting is to ask for feedback about the report's usability and value. The readers should give feedback about the availability and type of information needed, and the ease of use of the format (Rubin, 2008).

2.1.3 Usability Inspection Methods

Usability inspection is a generic term for a number of different methods: heuristic evaluation, guideline reviews, pluralistic walkthroughs, consistency inspections, standards inspections, cognitive walkthroughs, formal usability inspections and feature inspections. Holzinger (2005) lists a number of usability inspection and test methods, which are: heuristic evaluation, cognitive walkthrough, action analysis, thinking aloud, field observation and questionnaires. Even though all methods aim to find usability problems in user interfaces, they have different goals (Nielsen, 1994). The most common usability inspection methods are heuristic evaluation and cognitive walkthrough (Wania, Atwood, & McCain, 2006, Sharp et al., 2007). One of the objectives of usability inspection is to enhance the appreciation towards user-oriented perspective of user interface designers (Kahn & Prail in Nielsen, 1994).

The evaluation of user interfaces can be done in four ways: automatically, empirically, formally, and informally (Nielsen, 1994). Usability inspections belong to the latter way of evaluation. When usability inspection is carried out as an expert evaluation phase in user interface design process, usability and user experience goals and conceptual models can be considered as input attributes to evaluation process. The goal of usability inspection is to find usability problems in existing user interface design (Nielsen, 1994; Sharp et al., 2007). Also, it is recommended to recognize good features that improve usability in order to keep them in the further development. Nielsen (1994) presents that usability problems are used to make recommendations for fixing these problems and that inspection should take into account the classifying and counting the usability-related issues found in inspection. According to Sharp et al. (2007), the next step after evaluation is to identify needs and establish new requirements if usability-related issues are found. In both interpretations, the activities are the same, but the interpretation of Nielsen (1994) suggests that the recommendations are done during evaluation, but they also note that usability inspection belongs to a larger context, which in whole is called usability engineering life-cycle, by noting that "after the list of usability problems has been generated, a development team must redesign the user interface in order to fix as many of the problems as possible" (p. 3). Moreover, Nielsen (1994) presents that there are three major usages for usability inspection reports: to generate fixes and other redesign suggestions, use the list effectively by prioritizing issues by their severity, and finally, to estimate costs for implementing the suggested redesigns.

Usability inspectors can be usability specialists, but they can also be software development consultants with special expertise (e.g., knowledge of a particular interface style for graphical user interfaces), end users with content or task knowledge, or other types of professionals (Nielsen, 1994). In a usability inspection method-related study conducted with graduate students and usability experts respectively profiled as novices and experts, Chattratichart and Lindgaard (2008) concluded that the profiles should be defined more precisely as the education and working background did not determine well enough their performance whereas the researchers assume that the practical usability experience could be a better determinant.

Heuristic Evaluation

Heuristic evaluation is one of the two most commonly used usability inspection methods (Wania, Atwood, & McCain, 2006). According to Nielsen (1994), it is a so-called discount usability method because of being cheap, fast, and easy to use.

Heuristic evaluation is a usability inspection method utilized to find usability-related issues, both positive and negative. Sets of principles, also known as heuristics, are utilized to identify and categorize findings. The interfaces can be paper prototypes or user interfaces that have already been released. (Sharp et al., 2007.)

In addition to well-known Nielsen's (1994) heuristics, there are also updated variants of heuristics evaluation, HE+ and HE++ (Chattratichart & Lindgaard, 2008). According to a study by (Chattratichart & Lindgaard, 2008), HE+ and HE++ outperformed traditional HE in effectiveness. In addition, there are also heuristics for different types of software: for games (Pinelle, Wong & Stach, 2008) and for web 2.0 an extended framework by Thompson and Kemp (2009). Muller, Matheson, Page and Gallup (1998) added another inspection technique: participatory heuristic evaluation (PHE). Furthermore, user experience can be inspected using Arhippainen's (2009) heuristics.

Nielsen's (1994) heuristics include ten types of categories for usability-related findings: (1) visibility of system status, (2) match between system and the real world, (3) user control and freedom, (4) consistency and standards, (5) error prevention, (6) recognition

rather than recall, (7) flexibility and ease of use, (8) aesthetic and minimalist design (9) help user recognize, diagnose, and recover from errors, (10) help and documentation.

Nielsen (1994) provides also a five-point severity rating scale for assessing usability-related issues. The explanations for severities from the least severe to the most severe rating: not a usability problem at all, cosmetic problem only, minor usability problem, major usability problem, and usability catastrophe. It is suggested that the five numerical values should range from 0 to 4, respectively.

The inspection phase includes different practice possibilities with pros and cons: the inspection can be done by individuals separately, but also in a group. If inspections are done by individuals, they can inspect user interfaces using their own intuition without having other inspectors' opinions corrupt their perceptions. On the other hand, if the inspection is done in a group, other inspectors can aid other inspectors to gain more perspectives to the evaluation. Nevertheless, in both ways, the usability problems are discussed in a group before the actual reporting. It is suggested that evaluators should work independently, meaning that each evaluator work separately from other evaluators. By calculating the results of six projects, he concluded that one evaluator finds approximately 35% of possible usability problems. Thus, it is difficult for only one evaluator to find all the usability problems. Nevertheless, after each expert has evaluated the interfaces, a gathering should be organized to discuss the found usability problems in order to exclude false observations and to rate the problems. The end result of heuristic evaluation is a list of findings which is reported further to user interaction designers. (Nielsen, 1994.)

There are two approaches for evaluators to follow: freely or by following a specified user scenario. The benefit of following a properly chosen scenario is to evaluate how it is intended to be used, and on the contrary, when inspectors evaluate systems without a scenario, they might find important issues when the system is not used as intended. (Nielsen, 1994.)

Cognitive Walkthrough

The focus of cognitive walkthrough is on assessing only one usability attribute, ease of use, by exploration. This approach is close to a situation when no user training has been organized and one has to do all the learning by exploring features of a software product (Wharton, Rieman, Lewis & Polson, 1994). However, cognitive walkthrough has received criticism from Holzinger (2005) who argues that the process includes "possible tediousness and the danger of an inherent bias due to improper task selection, emphasis on low-level details, and non-involvement of the end user" (p. 73).

The actors in this usability inspection method are software developers, especially those who design user interfaces, usability experts, and peers of software developers, such as other designers and engineers or representatives from other units, for instance, marketing and documentation. (Wharton et al., 1994.)

Cognitive walkthrough consists of two main phases. The first phase is a preparatory phase which aims to set inputs to the second phase, respectively, the analysis phase. Together, the two phases are divided to five main steps: define inputs to the walkthrough, organize a meeting with analysts, the actual walkthrough, record critical information, and revise the interface to fix the problems (Wharton et al, 1994). Next, each step of the process is explained more widely.

The first step of the process is to define input to the walkthrough from four areas. The first area is to identify users and their prior experience or technical knowledge. The second area is to consider a sample task or a set of tasks. Multiple information sources can be used for task selection, such as marketing studies, needs analyses, concept testing and requirement analysis. If the system is very complex, the task suite should be limited to a reasonable collection of tasks. The third area involves creating a correct action sequence with descriptions for selected tasks. For instance, "Press the ENTER key" is one example of an action. The difficulty of an action should be similar to expected knowledge of end-users. The fourth and last area is to define the interface. The aim of this step is to describe the prompts of the interface that precede every action to complete the tasks successfully. In the early phase of the development, the definition of an interface can be a paper mock, and in the later phases, more accurate means can be used, such as response time and color distinctions. (Wharton et al, 1994.)

The second step is to settle the place and time for the walkthrough activity, and invite all participants to the gathering (Wharton et al, 1994). The actual walkthrough begins in the third step. The goal is to examine each action and tell why the targeted user group would choose the designed action and to tell a credible success story. The basis of stories are assumptions on user' prior knowledge and goals with an understanding of the user group's problem-solving process that helps a user to choose the correct action by guessing. Thus, the most essential features of the interface are those that provide links between task descriptions and correct actions, and on top of those, user's progress and previous actions should be visible (Wharton et al, 1994). Wharton *et al* (1994) present that the aforementioned issues are what the analysts should consider, and more specifically, they can be obtained by asking four questions:

1. Will the users try to achieve the right effect?
2. Will the user notice that the correct action is available?
3. Will the user associate the correct action with the effect trying to be achieved?
4. If the correct action is performed, will the user see that progress is being made toward solution of the task? (p. 112)

The fourth step of the cognitive walkthrough process, capturing critical information, is executed during the evaluation, meaning that the overall process does not proceed completely incrementally, but rather iteratively during the third and fourth step. An important matter regarding capturing information is to do it efficiently and effectively. A number of means are available for the purpose, for instance, videotaping and other material that is visible for the group, such as flip charts, overheads and paper forms. If necessary, the entire evaluation can be captured to verify or retrace communication. The most important information is user knowledge requirements, other presumptions considering users, changes in design, side issue notes and credible success stories. In addition, the following suggested key points should be captured: users' knowledge before performing the task and what is intended for the user to learn during performing the task. (Wharton et al, 1994.)

After the analysis phase is completed, the last phase of the process, revise the interface to fix the problems, begins. The goal of the step is to fix the issues gathered during the analysis. After fixing the interface another iteration of the whole process can be executed. (Wharton et al, 1994.)

2.1.4 Costs and Benefits of Usability Activities

Improving product's usability is good business because it "increases customer satisfaction and productivity, leads to customer trust and loyalty, and contributes to tangible cost savings and profitability" (Marcus, 2005, p. 17). User interaction design and development is always costly, and investing for it can create a bigger return of investment (Marcus, 2005). On the contrary, if no usability tests are done, users do the testing and might get angry (Donahue, 2001).

Many models exist to calculate the costs and benefits of usability focus (Rajanen & Iivari, 2007). Rajanen and Iivari (2007) found three categories of costs from five usability cost-benefit analysis models: one-time costs (Ehrlich & Rohn, 1994; Karat, 1994; Mayhew & Mantei, 1994; Donahue, 2001), recurring costs (Ehrlich & Rohn, 1994; Karat, 1994; Mayhew & Mantei, 1994; Bevan, 2000; Donahue, 2001) and redesign costs (Karat, 1994; Mayhew & Mantei, 1994). An example of a one-time cost is setting up a usability laboratory, usability professional's salary is a recurring cost, and redesign costs occur when a usability evaluation finds usability-related issues which are fixed in another iteration of user interaction design process (Rajanen & Iivari, 2007).

A broad definition of "a business benefit is a positive return that the development organization expects to obtain as a result of an investment" (Rajanen & Iivari, 2007, p. 3). Benefits of improved usability, i.e., return of investment (ROI) of usability can be divided in two different categories: benefits that drive internal or external ROI. Benefits of internal ROI, i.e, in-house products, are:

- Increased user productivity (Karat, 1994; Mayhew & Mantei, 1994; Donahuem 2001; Marcus, 2005)
- Decreased user errors (Mayhew & Mantei, 1994; Marcus, 2005)
- Decreased training costs (Mayhew & Mantei, 1994; Marcus, 2005)
- Reduced development costs (Mayhew & Mantei, 1994; Bevan, 2000; Donahue, 2001; Marcus, 2005)
- Decreased user support (Mayhew & Mantei, 1994; Dumas & Redish, 1999; Marcus, 2005)
- Decreased personnel cost through smaller staff turnover (Karat, 1994)
- Saved money in internal products (Dumas & Redish, 1999)

Vendor companies can possibly get a positive ROI from the following benefits:

- Increased sales (Ehrlich & Rohn, 1994; Karat, 1994; Mayhew & Mantei, 1994; Dumas & Redish, 1999; Bevan, 2000; Donahue, 2001; Marcus, 2005)
- Decreased customer support costs (Ehrlich & Rohn, 1994; Mayhew & Mantei, 1994; Dumas & Redish, 1999; Bevan, 2000; Donahue, 2001; Marcus, 2005)
- Reduced development costs (Ehrlich & Rohn, 1994; Mayhew & Mantei, 1994; Dumas & Redish, 1999; Bevan, 2000; Donahue, 2001; Marcus, 2005)
- Reduced maintenance costs (Donahue, 2001)
- Reduced training costs (Mayhew & Mantei, 1994; Dumas & Redish, 1999; Donahue, 2001; Marcus, 2005)
- Reduced documentation costs (Donahue, 2001)
- Increased perception of value of company by stakeholders (Marcus, 2005)
- Decreased personnel cost through smaller staff turnover (Karat, 1994)
- Improved reputation of the company (Dumas & Redish, 1999)
- Better notices in the media (Donahue, 2001)
- Competitive edge (Donahue, 2001)

- Advertising advantages (Donahue, 2001)
- Litigation deterrence (Donahue, 2001)

Rajanen and Iivari (2007) discuss the differences of five different usability cost-benefit analysis models. A model by Ehrlich and Rohn (1994) is focused on the aspect of vendor companies. The model includes three benefits: increased sales, reduced support costs and reduced development costs, and one-time and recurring costs (Rajanen & Iivari, 2007). The model by Bevan (2000) does not identify one-time costs, but the model includes recurring costs. According to the model, the benefits are: increased sales, reduced need for training, and improved productivity (Rajanen & Iivari, 2007). The third usability cost-benefit analysis model is made by Karat (1994), and uses human factors work as the viewpoint for the model. He identified increased sales, improved productivity and decreased personnel costs because of smaller staff turnover as the benefits of usability focus. The model identified costs from all three categories: one-time costs, recurring costs and prototype redesign costs (Rajanen & Iivari, 2007). The fourth model is made by Mayhew and Mantei (1994), and it identifies increased sales, decreased customer support, reduced number of changes in design, and reduced training costs as possible benefits. Also, they found examples of all categories of costs (Rajanen & Iivari, 2007). The fifth model is made by Donahue (2001) from the viewpoint of vendor companies. The model includes the following benefits: reduced development and maintenance costs, improved productivity and efficiency, reduced training costs, lower support costs, reduced documentation costs, litigation deterrence, increased e-commerce potential, competitive edge, advertising advantages, and better notices in the media. The model includes one-time costs and recurring costs (Donahue, 2001). Also a usability cost-benefit model for open source software development exists, but it will be discussed in the section considering usability in open source.

2.1.5 Core Human Computer Interaction Philosophies

Rajanen and Iivari (2013) summarized the core human computer interaction (HCI) philosophies. According to them, the philosophies are: usability specialists represent users, usability specialists "act in informative, consultative, participative and designer roles, sticking with the user focus, knowing the user, speaking for and fighting for the user in the development" and they have "decision-making power regarding important design solutions" (pp. 2-3).

The philosophies are supported by prior literature. For example, Cooper and Bowers (1995) mention that usability specialists are user representatives and according to Iivari (2006), they speak for the user in the development. Clemmensen (2004) categorize HCI specialists as "bridge builders", "designers", "analysts" and "programmers" (p. 809). However, Iivari (2006) argues whether usability specialist are designers by stating "it is yet a question mark whether they can be considered as designers as well also" (p.1). Nevertheless, Rajanen and Iivari (2013) note that usability specialists can receive merit "from knowing the users and their needs, as well as developing and evaluating the user interface design solutions" (p. 2). Also, usability inspection and testing practices include knowing the user population before evaluation processes are executed. After execution, usability testers gather user preferences and performance data for analysis and lastly, they develop recommendations based on user feedback and analysed results. The recommendations should be discussed with the development team (Dumas & Redish, 1999; Rubin, 2008). Furthermore, Rubin (2008) suggests to develop recommendations for better usability even if they are not in line with the company's politics.

2.2 Open Source Software Development

Open source software (OSS) refers to software with a freely available source code for everyone to access, read, modify, distribute and compile (Raymond, 1999) and a movement (Ljungberg, 2000). Ye and Kishida (2003) define OSS as "those systems that give users free access to and the right to modify their source code" (p. 1). Apart from having access to the code, the Open Source Initiative (OSI) list the ten principles, i.e, The Open Source Definition (OSD):

1. Free redistribution
2. Source code
3. Derived works
4. Integrity of the author's source code
5. No discrimination against persons or groups
6. No discrimination against fields of endeavour
7. Distribution of license
8. License must not be specific to a product
9. License must not restrict other software
10. License must be technology-neutral (http://opensource.org/osd)

However, the aforementioned principles cause confusion, as the OSD "replaced certain vague concepts in the Free Software Guidelines with some equally vague concepts about discrimination, authors' integrity, and software redistribution. Public discussions about license approval sometimes become arguments about what the OSD itself means" (Rosen, 2004, p. 6). Still, the principles form the basis for open licenses, such as: Apache License 2.0, BSD 2, BSD 3, GNU General Public License (GPL), GNU Library or "Lesser" General Public License (LGPL), and MIT license (http://opensource.org/licenses).

Some open source software products have been successful, such as Linux, Apache and Gnome (Shah, 2006; Zanetti, 2012), and thus, OSS is sometimes competitive with proprietary software (Mockus, Fielding & Herbsleb, 2002; Andreasen, Nielsen, Schrøder & Stage, 2006; Zanetti, 2012) and can "even outperform its commercial competitors" (Zanetti, 2012, p. 1587).

Open source is also an ideology with the emphasis to provide software free of charge. For some people, it is a way of life. One dimension of the ideology is concerned about the software's commercialism: there are anti-commercial people considering Microsoft as the evil enemy and the other end of the spectrum think that open source and commercial software can co-exist. (Ljungberg, 2000.)

Next, the following topics are discussed: OSS community structures, motivation of OSS developers, governance in OSS projects, and then usability-related topics: usability and usability bugs in OSS, and lastly, the core OSS philosophies.

2.2.1 Community Structures

Open source software communities are innovative communities (Shah, 2006). Oinas-Kukkonen and Oinas-Kukkonen (2013) describe open source communities being an example of networked improvement communities (NIC), a strategy used to enhance innovation creation. More specifically, open source communities are described as horizontal innovation networks (von Hippel, 2007). By enabling access to source code,

it is possible to create derivate works and meet personal needs while making natural product evolution possible (O'Reilly, 1999).

Loosely coupled open source communities consist of highly-skilled programmers who collaboratively develop software with hacker culture values. One of the values is that software should be free of charge (Ljungberg, 2000; Bergquist & Ljungberg, 2001). In contrast to traditional organisations, in most cases, open source software communities are open to everyone (Androutsellis-Theotokis, Spinellis, Kechagia & Gousios, 2010), thus, the communities are formed of volunteers, and for the success of the project, their motivation is an essential factor (Ye & Kishida, 2003).

In order for the community to sustain, it should attract developers. Different levels of involvement and status are described in an onion model which consists of four layers: core team, contributing developers, bug reporters and users. To create such a community is a key objective to sustainable software development communities (Aberdour, 2007). A more detailed description variety of roles in open source software communities from outer layers to the center are: passive user, reader, bug reporter, bug fixer, peripheral developer, active developer, core member and project leader (Nakakoji, Yarnamoto, Nishninaka, Kishida & Ye, 2002; Ye & Kishida, 2003; Ye & Fischer, 2007). However, these roles do not exist in all OSS communities as roles may vary from project to project (Nakakoji et al., 2002; Ye & Kishida, 2003). In addition, roles are not fixed, meaning that role transformation can exist in OSS communities because all members are potential developers. For instance, if bug fixers and peripheral developers are recognized in the community by making a number of contributions, they might eventually enter the team of core members. Also there might be evolution in the community structure and it might even change over time (Ye & Kishida, 2003). The closer to the center, the more influence a community member has. Passive users only use the system similarly to the use of proprietary software. Readers use the system actively and read the source code in order to understand the system. Bug reporters find and report bugs, but do not fix them. However, not all bug reporters read the source code. Bug fixers create patches for the bugs which have been found by themselves or other community members. They have to read the part of the code where the bug is located. Peripheral developers contribute to the system irregularly in short periods. Active developers contribute code to the project regularly. Core members have the responsibility to coordinate and guide development. They have earned their status by making significant contributions to the project over a long time span. Project leaders have often started the project and thus, they guide the overall development of the project (Ye & Kishida, 2003; Ye & Fischer, 2007). However, the leader might not have a clear picture for the project in the beginning, and thus, the OSS community drive the evolution of the system by collaborating (Nakakoji et al., 2002).

Oezbek, Prechelt and Thiel (2010) studied the structure and e-mail communications of OSS communities and noted that they have a tightly integrated core, co-developers were loosely collected with a strong tie to the core, and peripheral developers were connected only to the core or not connected to any community layer showing low peer-to-peer assistance. Thus, they argued that the so-called onion model is not the most descriptive metaphor to describe the structure of OSS communities, and thus, they suggested a model called earth, moon and stars. The core refers to earth, co-developers form the moon and peripheral developers the dot-like stars.

2.2.2 Motivation of Developers

OSS developers have "a personal scratch to itch" meaning the cause why open source software projects begin referring to a certain need for which they try implement a solution (Raymond, 1999). Because the access to the project's source code is available for free, it sometimes attracts people with similar problem, and thus, some users start to collaborate and become co-developers. Eventually, they create a community for the software project (Nakakoji et al., 2002; Ye & Kishida, 2003). The motivation of volunteers to participate and contribute is essential for the success of the project (Ye & Kishida, 2003). However, "[the] majority of participants leave the community once their needs are met" (Shah, 2006, p. 1000): free-riders, i.e, people who let others write the code, but use the product, exist in open source communities (Ljungberg, 2000). In cases of the ones who stay, participation might become a hobby as the motive changes in time (Shah, 2006). In general, OSS projects are dependent on volunteers (Shah, 2006). However, not all OSS contributors are volunteers as currently there is a number of OSS project which are sponsored and supported by commercial companies who might even use their own developers for OSS development (Mockus, Fielding & Herbsleb, 2002). In fact, Lakhani and Wolf (2005) observed that paid contributors work more hours.

Often, developers working on OSS software can freely choose their assignments based on their real passion, and thus, they write code with care and creativity (Raymond, 1999; Shah, 2006). They are also proud and responsible to deliver high-quality code "because peers they truly respect will review their efforts" (Fitzgerald, 2011, p. 4). Moreover, programmers hope to receive reward and recognition in order to improve their status in the community by delivering high-quality code (Raymond, 1999; Aberdour, 2007), knowledge and solutions (Ljungberg, 2000). "Producing high-quality source code is the key motivator for members when understanding the open source community as a social actor" (Bergquist & Ljungberg, 2001, p. 319). Ljunberg (2000) note that, in a gift culture, social status is defined by what you give away, which is a way to get power and control. Furthermore, programmers want their name to appear in the contributors list (Kishida & Ye, 2003), and some want to be recognized by contributing to a successful project (Bergquist & Ljungberg, 2001). As developers contribute to communities, they have a possibility to earn new roles (Nakakoji et al., 2002). Moreover, Ye and Kishida (2003) argue that learning, an intrinsic motivation, is also another motivational factor for people to participate to OSS communities.

In Mozilla's case, improved documentation, tutorials and improved tools and processes increased participation to the project (Mockus, Fielding & Herbsleb, 2002). New challenges, and an opportunity to learn new technologies and tools are also key motivators (Kishida & Ye, 2003).

Two types of participants can be identified: need-driven participants and hobbyist participants. Both types can be found in open source communities, but in gated source communities, only need-driven participant can be found. In case of need-driven participants, need is the base for choosing a software and creating code for it. Need-driven participants have a specific need, for example, for work purposes, and they choose the software accordingly and contribute to the project to fit their needs. The motives for need-driven participants are reciprocity, future improvements, source code commits and career concerns. In more detail, a need-driven participant helps others as the community members have helped him or her, the participant want to get feedback and express his or her needs, contribute to the code base to fit his or her needs, to get reputation and new skill that will help to a new job. However, career concerns were found to have a relative little importance as a motive. The other type of participation is hobbyist participation, which is done for fun and to get a challenging hobby-like

activity. Some hobbyist participate to have freedom and creativity in contrast to their daily work. A motive for hobbyist participants is to get feedback for others. (Shah, 2006.)

There are three types of motivations: enjoyment-based intrinsic motivations, obligation/community-based intrinsic motivations and extrinsic motivations (Lakhani & Wolf, 2005). Based on a questionnaire survey by Lakhani and Wolf (2005), they found all types of motivations from open source software developers. The top most occurring reason (44.9%) is creativity, an intrinsic type of motivation, as "project code is intellectually stimulating to write" (p. 12). The second most frequent reason why people contribute is to improve programming skills (41.8%). About one-third of the respondents answered that source code should be open. 28.6 percent of the respondents felt that they had to give back to the community in return of providing the software. One-fifth of the respondents felt that working with the project team was motivating. Community reputation, professional status and beating closed-source software had relatively low impact on developers' motivation, on the contrary to the beliefs of prior literature. Moreover, respondents felt that being part of a hacker community is motivating: 42% strongly agreed and 41% somewhat agreed (Lakhani & Wolf, 2005).

According to a questionnaire survey by Andreasen et al. (2006) with 24 respondents, 88% of the population answered that their motivation to contribute to OSS projects was to strengthen free software and that contributing is intellectually stimulating. 75% wanted to improve their skills, 54% answered community reputation, one-third contribute because of professional status, and only one responded was paid by employer. Three respondents answered other reasons for their motivation.

Lanamäki, Rajanen, Öörni and Iivari (2015) proposed a gateway theory of online participation. The proposed theory has similarities to the gateway theories of drug usage as in both theories a person steps over a certain line and becomes sensitized to the next step. The gateway theory of online participation is divided to two parts. In the first part, the participation involves "uncertainty, involving trial and error, unknown risks and rewards, and the availability of technology-facilitated services" (p. 1). In the second part, a person performs different activities which gives him or her awareness of new opportunities. The theory was tested by interviewing two persons: a Wikipedia contributor and a core team member of a FLOSS game. Both of them got involved to the IT solutions by accident, and after making the initial contributions, both of them got "hooked" to the communities and made more contributions

2.2.3 Governance in OSS

As OSS development is done by a geographically distributed team, more focus is required in coordinating tasks (Aberdour, 2007). Usually OSS development has a central person or body who chooses the developed code for official releases (Mockus, Fielding & Herbsleb, 2002). The core team, consisting of experts, does most of the coding work and coordinate work informally knowing other core developers' skills and bug fixers and testers help them in the development. However, creative chaos can exist as developers select tasks themselves (Fitzgerald, 2011). The main communication methods in OSS communities are the web and e-mail (Ljungberg, 2000; Zanetti, 2012). According to O'Reilly (1999), "a community needs to develop processes for voting on new features, deciding who has access to the source tree, and communicating in ways that do not stifle the free-floating development that is so central to the appeal of open source" (p. 36). Androutsellis-Theotokis and his associates (2010) divide governance models to two categories: monarchical and federal.

A common coordination model in open source projects is so-called benevolent dictatorship which is usually established after the initiator of the project has attracted other contributors to participate to the project (Ljungberg, 2000; Andreasen et al., 2006; Fogel, 2013). Even though the term is benevolent dictator, other interpretations are "community-approved arbitrator" or "judge" (Fogel, 2013, p. 73). Benevolent dictatorship is a hierarchical model, a variation of monarchical model. An example of the model is Linux project in which Linus Torvalds is the dictator and parts of the authority is divided to core team members (Androutsellis-Theotokis et al., 2010). In the model, dictators do not make all the decisions, but the dictator has the right to make final decisions. Dictators participate to discussions and when the community does not reach consensus, a dictator makes the final decision. Also, they do not have to be experts in all fields, and dictators can even make mistakes, however, they must have a sense for overall design (Fogel, 2013). Another option to have a single dictator is to have a rotating dictatorship (Ljungberg, 2000).

Projects tend to move towards more democratic systems when they get older (Fogel, 2013). Democratically distributed models are federal leadership models (Androutsellis-Theotokis et al., 2010). Often, projects have a written constitution and the mechanism to reach democracy are basically a clear consensus or voting. In these kind of cases, consensus means, for instance, that discussions lead to common practices, and version control systems can be used to choose the best option at code-level as they provide an ability to revert commits. Voting is the last option to rely on when consensus cannot be reached. It is recommended to include an option to write down alternative options, which can be discussed further. Also, projects should have a rule who have a right to vote (Fogel, 2013).

Another decision-making model in OSS communities is meritocracy. Projects leaders, i.e., core team members are selected based on their merits through contributions following a meritocratic model as developers have a chance to rise from bottom to up (Bonaccorsi & Rossi, 2003). Contributors build trust through merits, and in consequence, receive more influence in development (Frisberg et al., 2002; Andreasen et al., 2006).

2.2.4 Usability in Open Source Software Development

Critics claim that it is impossible to predict the usability of an open source software product (Fitzgerald, 2011) or that it is poor (Nichols & Twidale, 2003; Hall, 2014) and that OSS products have "little or no emphasis on usability" (Andreasen et al., 2006, p. 303) as achieving good usability is not the primary objective in OSS products' goals (Çetin & Göktürk, 2008). Hall (2014) notes that the source of usability problems in OSS is cultural. OSS is developed and chosen by tech savvy people causing problems for technologically less aware users (Nichols & Twidale, 2003; Fitzgerald & Ågerfalk, 2005). Nadeau (1999) notes that Microsoft takes into account its "most ignorant users", and on the contrary, Linux and OS/2 developers "listen to their smartest customers". According to Fitzgerald and Ågerfalk (2005) that might be the source problem why OSS interfaces might not be so user-friendly. Çetin and Göktürk (2008) note that F/OSS developers have a tendency to create feature-centric projects instead of applying user-centered design (UCD). Usability experts do not participate much to OSS development. The possible explanations for this are that the ratio between usability experts and hackers favor is higher on hackers' side, there is no interest or incentives for usability experts to contribute or they do not feel to be welcomed, and there are not so good career opportunities for usability experts in OSS (Nichols & Twidale, 2003). Similar kind of observations were mentioned by Rajanen and Iivari (2013) who note that

communities might not recognize reports, recommendations and mock-ups as contributions, and usability specialists have to contribute non-usability activities in order to gain merits.

However, the user-centered design movement has been perceived to close the gap between users and programmers with its techniques as OSS catches up commercial software development in terms of following the interface ideas made by commercial development (Nichols & Twidale, 2003). The developers who made Linux were nearly the only users of the operating system for many years, and thus, as the desktop environments will be used by a wider audience, the OSS community is starting to understand that they are not their only users (Frishberg et al., 2002). Futhermore, Raza, Capretz and Ahmed (2012) present that developers are not the only user of OSS anymore. Frishberg and his associates (2002) have a few suggestions that the OSS could benefit from: creating systems that can be used by non-programmers, consistent behavior, and respecting the needs of disabled users. The users include elderly, children and people with disabilities. Thus, there is a need to improve help features to be more interactive and dynamic (Raza, Capretz and Ahmed, 2012). Nichols and Twidale (2003) list eight approaches which could improve OSS usability: commercial approaches, technological approaches, academic involvement, involving the end users, creating a usability discussion infrastructure, fragmenting usability analysis and design, involving the experts, and lastly, education and evangelism.

Rajanen, Iivari and Anttila (2011) examined how to introduce usability activities into OSS development using four cases. A participate approach was used in two cases and in the other two cases, the approach was consultative. The conclusion of the study was that usability introducers should choose a participative approach rather than consultative. Other factors that could help to be more successful in usability activities are: to understand OSS development philosophy, principles, and characteristic, establish a dialogue with the core developers and ally with them, identify the possible benefits of improved usability and use them in argumentation, maintain an objective view when adapting usability activities to the development, inform core developers and community about usability activities and user interface improvement suggestions, execute usability activities for the correct version of the product at correct development phase, and promote the interests of non-technical users.

Rajanen and Iivari (2010) found a gap in the existing usability cost-benefit analyses when the development context is open source software development. There are similarities and differences when compared to usability cost-benefit analysis models in commercial software development context. The researchers derived a model for OSS context from existing commercial models and added their own observations through interviews from two cases which were community and company OSS development projects. The implications of usability benefits were the following: similarly to increased sales in commercial development, community-based open source software had “more users” and “happy users”, whereas company-supported open source software had “increased user satisfaction” and “good UI as important image and competitive factor“. Implications for reduced development costs in both cases were that “community takes active part in providing feedback and redesigning the solution in the forums”. Separately, community-based software had “early releases give more time for redesign”, and the observation from company-supported was that “early releases give more time for usability work and feedback, help to identify needs for redesign early”. In the last benefit category called reduced training and support costs, both cases had peer support in forums. The identified one-time costs for community open source software was setting up a usability forum, and for company open source software, “establishment of a usability laboratory, guidelines [and] company tailored usability methods”. The found

recurring costs for community-based software was that “if usability specialists work in the projects their time and effort is needed”, and in company-supported project usability specialists were hired. Two similar redesign costs were found in both cases: “uncontrollable continuous redesign” and “users and developers produce and comment on mock-ups”. Furthermore, “potentially a lot of time spent in redesign” was identified solely in the community case, and “potentially increasing redesign costs since there is more time and opportunities for redesign than in closed source development” was identified in the company-supported project (p. 8).

Hedberg, Iivari, Rajanen and Harjumaa (2007) reviewed existing research literature about general quality and usability in OSS development. According to the article, the current usability practices in OSS development include that it is assumed by the development that users, who are also co-developers, report bugs. The current research situation at that time was that only a few articles were published and the articles had defined usability and UCD very vaguely. The article recommends a usability practice based on the ISO definition of usability and UCD. Lastly, the authors make a research recommendation on how to adapt UCD practices and principles for OSS development.

Hedberg and Iivari (2009) proposed a model in which HCI specialist integrate to OSS communities. The model, focusing on technical-oriented and human-oriented thinking, is based on usability engineering and UCD practices and takes HCI specialists into account in projects. In the model another onion-like level, called human level, is added to the traditional onion model structure of OSS communities, called technical level. Also, the languages used in both levels are different. The HCI specialist are located in the upper level including different HCI roles such as usability specialists and usability evaluators. The interaction between the levels is made through decisions. The model was evaluated through a project and feedback was received from approximately 100 experts and enthusiasts in an OSS seminar.

Andreasen and his associates (2006) made an empirical study of OSS developers’ opinions about usability and usability practices in OSS projects. The questionnaire survey included 24 developers. 83% of the respondents answered that the importance of usability is high, very high or extremely high. One interviewee told that doing usability-related activities is not intellectually stimulating, another interviewee told that the goal of his OSS is not to reach wider audiences. Also, the researchers asked the participants their definition for usability, and the result was that they had very simplistic views of the term and there was a lot of different definitions for it. Also Terry, Kay and Lafreniere (2010) made an interview survey about usability perceptions of OSS community members resulting to wide range of definitions of usability, but the researchers noted that as a whole, the respondents had a clear view about usability. In the study by Andreasen et al. (2006), the respondents felt also that including a usability expert to the project would reduce the democratic nature of decision-making and preferred the independent group approach. In addition, another answer was that involving a usability expert would be difficult because ”OSS people don’t like too much to be told what to program” (p. 308). Moreover, the respondents had different views on which stage of development usability should be applied, but still, half of the respondents answered that it should be done in the beginning. The interviewees told some reasons why the suggestions of usability experts are ignored: one reason is that some suggestions are too big to be implemented because of the underlying code, and due to the nature of OSS as a distributed development in which developers rarely meet face-to-face, it is difficult to build trust to usability experts as “it can be difficult for a usability expert to show merits, since usability improvements are more difficult to measure than the programming of a new feature” (p. 309). However, the study reported a case in which the trust was established resulting to a direct contact between the developers and

usability experts. This case became “very gratifying” (p. 309). The study showed also that instead of usability evaluations, guidelines were used. However, 42% of the respondents answered that they had conducted expert inspections, but often without usability professionals. According to the interviewed usability professionals, guidelines should not replace usability evaluations. Only 8% had used a usability laboratory. The study indicated that lack of money or difficulty to rent a proper laboratory are some reason for not doing laboratory tests. According to a survey by Terry, Kay and Lafreniere (2010), the respondents, as a whole, had used 18 different usability practices in their projects. The five most popular practices in frequency order were:

1) discussion on IRC and mailing lists
2) feedback on mock-ups, prototypes, and custom builds
3) drawing up specifications, setting milestones or articulating visions
4) involving UX people
5) usage of human interface guidelines

Sun Microsystems, a commercial company, has contributed HCI expertise to OSS projects such as GNOME, OpenOffice.org and NetBeans. They found that the decentralized nature and engineering-driven approach of OSS projects are in conflict with processes used in corporations and usability engineering methodologies. Sun Microsystems has contributed to the aforementioned projects, for instance, by giving human resources, establishing a user interface specification document, graphics, a usability study which resulted to establishing a usability group by the community, interface guidelines, designed an accessibility framework, user documentation and product requirements (Benson, Müller-Prove & Mzourek, 2004). It is noted that the challenges usability professionals face are unclear user profiles, unclear ownership of the code, processes, attitude (Benson, Müller-Prove & Mzourek, 2004) and communication (Benson, Müller-Prove & Mzourek, 2004; Çetin & Göktürk, 2008). To solve the communication problem, Çetin and Göktürk (2008) suggest to use a content management system (CMS) and communication tools for project members in order to enhance cooperation and information flow. Moreover, a user profile, guidelines and critical user tasks list should be defined and to provide an adequate level of visibility for project members.

According to a questionnaire survey by Çetin and Göktürk (2008) examining the reasons why high-level usability is not achieved in OSS projects revealed four reasons. Firstly, developers are ignored about contextual inquiry and UCD process. As a result, they do not have the knowledge how the product is used. Secondly, usability experts lack knowledge of how to act in OSS projects: they criticize rather than contribute code. Thirdly, developers do not gain of working with usability people and often ignore user feedback when the product does not work as developer has expected. Fourthly, the research in the field is incomplete resulting to a lack of respect. Also, there is a lack of scientific usability metrics that developers can relate making it difficult to assess user interfaces.

Raza, Capretz and Ahmed (2011) investigated the influence of key factors on OSS usability and concluded that “users’ expectations, usability bug reporting and fixing, interactive help features and usability learning have a positive impact on usability of OSS projects” (p. 119). Also, they tested whether usability guidelines had any significance on OSS usability and concluded that it was not statistically significant. A year later, Raza, Capretz and Ahmed (2012) analyzed the link between usability issues in open source software and support given through online forums. They found out that forums help to identify and fix usability issues. Still, in only 10.95% of the analyzed projects, at least some focus was put on usability issues. Alongside forums, Terry, Kay

and Lafreniere (2010) found out that IRC and mailing lists are also popular in discovering and discussing usability issues between users and developers.

A study by Rajanen, Iivari and Keskitalo (2012) discussed the impacts of four student projects contributing usability activities into an open source game project using a participative approach. The usability activities included usability testing, heuristic evaluation and cognitive walkthrough. A member of the first project was already a community member of the project with a usability champion status, and acted as an informant for the core team. The student team developed improvement suggestions and code patches. Later, the practice of communicating results through a usability champion instead of direct communication with the core team was found problematic. However, the usability activities proved to be successful as they clearly had an impact to the game by taking part to discussions, reporting and submitting bugs and editing the community wiki pages. In addition, one project member received commit rights to the project's version control system by gaining trust and merits through active discussion. The researchers concluded that such kind of participative approach is beneficial as usability experts work as a part of the development team. Thus, other usability experts may use the same approach when starting to contribute usability activities to software projects. Furthermore, the core team did not care that the contributors were students, but on the contrary, thanked them of the high level quality of the reports indicating that the achieved merits were more important than their status. However, the results are derived from only one study, and thus, cannot be generalized, which creates a need to conduct similar kind of studies.

Rajanen and Iivari (2015a) examined usability work and culture in open source software context using data collected from four cases in which students contributed usability activities to open source software projects. They found contradictions from the existing literature: some sources recommend that the engineering or organizational culture should be adapted to comply with usability work whereas other sources suggested that usability work should be modified to fit the engineering or organizational culture. The main types of cultures are: group culture, adhocracy culture, hierarchical culture and rational culture. The target communities used in the study were identified by the values the culture types emphasize. The researchers identified the culture types of the four target communities as adhocratic, group or hierarchical types. The only project that was considered successful had adhocratic type of culture, and thus, the findings of the study suggest that the most suitable type of culture for usability work in open source software context is adhocracy.

Even though OSS development should be open to everyone to participate, gatekeeping mechanisms exist. According to study conducted by Rajanen, Iivari and Lanamäki (2015), usability projects in OSS context were gatekeeped using three tactics: non-response, social exclusion and false acceptance. By using six projects as research cases, they found out that non-response tactic was used in three of them, three projects encountered social exclusion, and one received false acceptance.

2.2.5 Usability Bugs in OSS

As discussed earlier, bug reporters form one the layer in the so-called onion model (Aberdour, 2007). Bug reporters do not have to be able to code, as "finding and reporting bugs does not involve code changes" (Fitzgerald, 2011, p. 28). The developers who fix bugs belong either to the peripheral or active developer layer in the onion model (Ye & Kishida, 2003), and often, the developers choose their tasks as the issue management does not have mechanism to coordinate development, use project plans,

schedules, delivery lists (Mockus, Fielding & Herbsleb, 2002). Usability bugs is a non-functional of type of a bug. According to Nichols and Twidale (2006), "usability bugs may not be afforded the same status as functionality bugs, particularly those latter that lead to crashes" (p. 4). Wilson and Coyne (2001) point out that usability bugs cover 20 to 50 percent of all bugs, and they might even need a separate database from functional bugs.

According to a questionnaire survey by Çetin and Göktürk (2008), 66.1% of developers and usability experts gain benefits from using a bug reporting tool called Bugzilla, but still only 23.8% had a particular place for usability issues in the bug reporting tool. On the other hand, 84.2% of the respondents account usability bugs for real bugs. Based on these outcomes, the researchers conclude that successful OSS projects need proper bug tracker configuration for reporting usability bugs.

As discussion is open in open source software development, Twidale and Nichols (2005) explored usability-related discussions. They read usability-related bug reports of OSS projects and characterised how they were addressed of and resolved. The researchers observed that a bug reporting tool called Bugzilla was very text-centric with an ability to attach screenshots, but sometimes, due to protect reporters' privacy, a feature to support blurring should be implemented. In addition, the limited expertise or absence of usability experts creates a challenge to advocate end users. Another problem is to classify bugs as often a duplicate identification tool is not used, and classifying usability bugs is equally difficult than functional bugs. Furthermore, the lifecycle of a bug to be implemented to production is a complex process, and as the comments are linearly displayed, the researchers suggest to use threading in bug reports.

2.2.6 Core OSS Philosophies

Rajanen and Iivari (2013) summarized the core philosophies of open source software using several sources, and many of them have been used earlier in this OSS literature review section. The core OSS philosophies according to Rajanen and Iivari (2013) are: software as communal resource, voluntary collaboration, loosely coupled community interaction, different levels of involvement and status, gaining merit and reputation through contributing to the community, especially high quality code, and "scratching your own itch".

Software as communal resource is one of the core values of the open source and free software communities (Rolandsson, Bergquist & Ljungberg, 2009). In general, communities are loosely coupled communities respecting the hacker culture values (Ljungberg, 2000). Shah (2006) observed that developers are mostly volunteers, and when they have met their personal needs, the majority leave the project. OSS communities have different levels of involvement and status which are depicted in so-called onion models (Yarnamoto et al., 2002; Ye & Kishida, 2003; Aberdour, 2007) and in a model called earth, moon and stars (Oezbek, Prechelt and Thiel, 2010). Even though all the roles are not in every project, the roles in open source software communities from outer layers to the centre are: passive user, reader, bug reporter, bug fixer, peripheral developer, active developer, core member and project leader (Nakakoji et al., 2002; Ye & Kishida, 2003; Ye & Fischer, 2007). Also related to different level of involvement and status and gaining merit and reputation, Mockus, Fielding & Herbsleb (2002) note that developers who have contributed good quality of code have chances to gain commit access to OSS project's version control system. The definition of merit is "the quality of being particularly good or worthy, especially so as to deserve praise or reward" and reputation is defined as "the beliefs or opinions that are generally held

about someone or something" (MOT Oxford Dictionary of English). In this thesis, reputation is considered as a positive belief or opinion. In addition, getting to an inner layer of the so-called onion model is a "reward and recognition for each member's abilities and achievements" (Aberdour, 2007, p. 59). Lastly, OSS developers start their own projects when they have a need, i.e., "an itch", for a specific software, i.e., "a scratch for the itch", and thus Raymond (1999) describe that when OSS projects start, developers are "Scratching your own itch". Also, Aberdour (2007) notes that developers choose tasks from the project's roadmap based on their "itch" which they need to "scratch". In general "scratching your own itch" means that a developer does something in which he or she has a personal motivation (Raymond, 1999).

3. Material and Methods

The research material is derived from an open source software project called Concrete5 and a usability project called UKKOSS11 which introduced usability activities into the open source software project. The usability project belongs to a larger research program which has several publications (e.g., Rajanen & Iivari, 2015a; Rajanen & Iivari, 2015b; Lanamäki et al., 2015) and it was conducted in three parts from 2013 to 2015.

The first part started in the beginning of 2013 with a kick-off meeting via email correspondence with one of the project leaders of the community, and finished in December 2013. The usability team consisted of two junior researchers. The first contributions consisted of conducting heuristic evaluation, cognitive walkthrough, usability tests for system version 5.6.1.2, and a How To document was written for the community on how to conduct usability tests and inspections for the system, but it was never published. The results, both positive and negative, of the usability test and inspection were sent to the aforementioned project leader of the community.

The second part of the project was done during the spring term in 2014 with the same team. After the first part of the project leader asked the team if whether they could conduct usability activities for version 5.7 of the system which was under development with renewed user interfaces. The usability team conducted heuristic evaluation and cognitive walkthrough inspection, and submitted the results to the project leader.

The third part of the project was done during spring term 2015 for system version 5.7.3.1 with a team consisting of five members, one of them was the same as in the prior phases being the key communicator with the community. The conducted usability activities in this part included heuristic evaluation and usability tests.

Concrete5 is an open source content management system (CMS) having an active community. The project has a core team and a group of active developers plus other developers that are loosely coupled to the community. The main communication methods of the community are community's website including forums, GitHub issue tracker, IRC channel and the core team uses Slack. The development of the software began in 2003 and the first release of the software was in 2010 under MIT software license. The design objective of Concrete5 is ease of use of users with minimum technical skills.

Case study research is one of the research methods in social science. It should be used when "the research questions are *how* or *why* questions", "a researcher has little or no control over behavioural events" and "the focus of study is contemporary" (Yin, 2014, p.2). One of the definitions of case study research is that it "investigates a contemporary phenomenon (the "case") in its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident" (Yin, 2014, p.2). History and experiment research methods also attempt to answer to *why* questions, but experiment research requires control of behavioural events and history research do not focus on contemporary events (Yin, 2014). Another definition is "the essence of a case study, the central tendency among all types of case study, is that it tries to illuminate a decision or set of decisions: why they were taken, how they were implemented, and with what result" (Schramm, 1971, as cited in Yin, 2014).

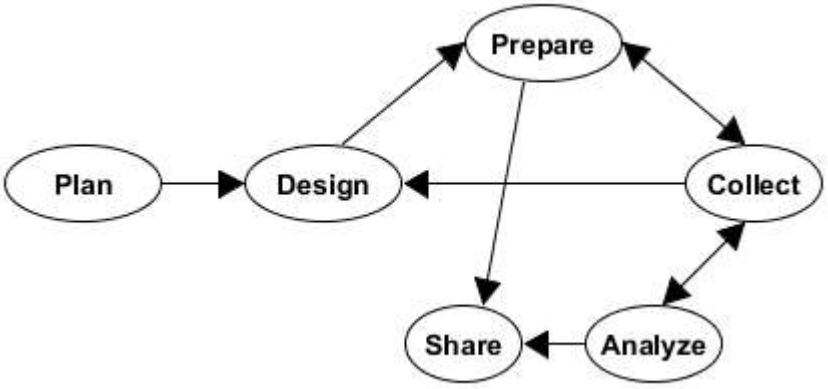

Figure 2. Case study research process. Adapted from Yin (2014).

The main phases of case study research method are planning, designing, preparing to collect evidence, collect evidence, analysing the evidence and reporting (Yin, 2014). The flow of the process is linear, but might have some iterations. Figure 2 visualizes the research process.

3.1 Research Plan

First, a thorough literature review should be made. Then, the research questions and objectives should be posed carefully and thoughtfully. Next, the research method is chosen. The form of research question(s) help to guide to choose the correct method alongside the extent of control a researcher has and the degree of focus on contemporary events. (Yin, 2014.)

There are five concerns when doing case study research properly: "conducting the research rigorously, avoiding confusion with teaching cases, knowing how to arrive at generalized conclusions if desired, carefully managing the level of effort, and understanding the comparative advantage of case study research" (Yin, 2014, p.2).

The first question examines the impacts of the project to the OSS community. One approach to answer the question is to explore the usability-related discussion in the community similarly to Twidale and Nichols (2005) who explored usability discussion in open source software development by analyzing the kind of discourse qualitatively. Even more related to this study, Rajanen, Iivari and Anttila (2011) examined different approaches in introducing usability activities into open source software development with an attempt to figure out which approach is suitable. According to Schaffer (2004), the institutionalization of usability usually requires a so-called wake-up call.

The first research question is:

1. *What kind of impacts usability evaluations had on the community?*

Because the second and third research questions consider the existence of core human computer interaction and open source philosophies similarly to a research conducted by Rajanen and Iivari (2013), the same research method, case study research (Yin, 2014), will be used.

The second and third research questions are:

What aspects of HCI philosophy existed during UKKOSS11 project?
What aspects of OSS philosophy existed during UKKOSS11 project?

3.2 Research Design

Research design describes a link from the data to be collected to the research questions. The design includes theoretical propositions. Also, it defines "the case or unit of analyses" (Yin, 2014, p. 27). In addition, this phase of research defines "what questions to study, what data are relevant, what data to collect, and how to analyze the result" (Yin, 2014, p. 29).

In interpretative case studies, there are two roles for researchers: outside observer or involved researcher, respectively, through participant observation or action research. Neither of the roles can be accounted for objective reporters as the researcher's subjectivity is involved in data collection and analysis. The outside observer's role requires to take some distance from the observed organization and thus, the personnel do not consider the researcher as one them. The advantage of using outside observer's role is that the researcher does not have direct impact to interpretations and outcomes, however, the disadvantage is that the researcher is not involved in as many occasions as possible and therefore, the information from the inside is limited. In participant observer's role, the researcher is a temporary member of the field group. The advantage of this approach is to get inside view and better access to confidential or sensitive issues. The disadvantage is that the researchers might have more direct influence in views and activities (Walsham, 1995). The level of involvement can be viewed as a spectrum because it might change over time (Walsham, 2006).

There are five important design components in case study research: questions, propositions, unit(s) of analysis, logic linking the data to propositions, and criteria for interpreting the findings. Research questions guide to select the correct research method. To form research questions, first, it is suggested to use the prior literature to narrow interests to a key topic or two. Then a few relevant studies should be read to look out for conclusions or research gaps to stimulate to in forming the research questions, and lastly, another set of studies on the same topic should be read. Propositions help to put the focus on what to examine within the scope of the study which helps to go to the right direction (Yin, 2014). Walsham (1995) mentions that theories can be used to guide design and data collection or to guide iterative process of data collection an analysis. The reason to use theory early in an interpretative case study is to create a preliminary framework addressing prior knowledge, topic and an approach of earlier empirical work. However, there is a danger of using a theory as a researcher might not see other important issues outside the theory. According to Yin (2014) Unit of analysis means to actual case in a case study. The case and its boundaries should be defined. A case can be a specific event or an entity other than a single individual such as small groups, communities, decisions, programs or organizational change. Linking data to propositions is related to data analysis step of case study research process and it includes techniques such as pattern matching, explanation building, time-series analysis, logic models and cross-case synthesis. The techniques will be explained more widely in the analysis chapter. Another important issue regarding to linking the data is the amount of collected data: one has to be cautious to not to collect too much data or too little. Criteria for interpreting the findings is the last design component. In quantitative research, a p level of less than .05 is considered statistically significant as thus it is considered as criteria. However, as case studies usually rely on qualitative data and thus,

for example, rival explanations can be used and if more rival explanations are addressed and rejected the findings become stronger.

The quality of the design can be tested by its construct validity, internal validity, external validity and reliability. The tactics to reach construct validity are to have multiple sources of evidence, create a chain of evidence and that key informants review the draft of the reports. The methods to increase internal validity are to conduct pattern matching and explanation building, address rival explanations and using logic models. In order to have external validity, the following tactics should be used: using theory in single-case studies and replication logic in multiple-case studies. Lastly, reliability can be achieved by using case study protocol and by developing case study database. (Yin, 2014.)

Related to the first research question, a research by Rajanen, Iivari and Anttila (2011) studied the suitable approach for introducing usability activities into open source software development concluded to mixed results. However, they noticed that participate approach might be more suitable than consultative. The logic for the first question is derived from Schaffer's (2004) process to institutionalize usability into a project: first a "wake-up" call is made, and the institutionalizing follows. Also, a prior study has been made regarding to the second and third research questions with four cases, which yielded also mixed results (Rajanen & Iivari, 2013). The logic of linking the data to propositions and the criteria for interpreting the result can be derived from the prior researches mentioned above.

In this study, the cases are UKKOSS11 usability project and Concrete5 project. The approach used in the study was involved researcher and the data was collected from the correspondence with the community. Also two questionnaires were sent to a project leader prior to and after the project, and as the target system has an open source community, it has publicly available information, for instance, in its website, forums, IRC channel and version control system.

3.3 Prepare

Prepare is the third phase of the case study research process. The preparations for conducting a case study include training case study research skills, especially for a specific case study, developing a protocol for the study, screening candidates and selecting cases, conducting the study, and gaining approval for human subjects protection. The needed skills in this phase are: to ask good questions, being a good listener, staying adaptive, having a firm grasp, avoiding biases and conducting research ethically. The case study protocol includes four sections: an overview of the case study, data collection procedures, data collection questions, and an outline of the case study report (Yin, 2014).

The training for conducting the study was made by reading the existing literature about research methodology and prior researches related to research objectives. The candidates used in the study were people related to UKKOSS11 project including the project members and a project leader who was the contact person in the open source community. Other people in this study are community members who discussed in the publicly available community forum. In this thesis, the real names were changed to their roles to protect their real identities.

The protocol in this study was the following. The selected case in this research was the UKKOSS11 usability project and Concrete5 open source software project. The decision to make a master's thesis research was made after the first phase of UKKOSS11 project.

It was decided to do a case study similarly to the prior research related to introducing usability activities into open source software development, and two more sets of usability activities were contributed after the decision. The data collection procedures included asking questions (Appendix A & Appendix B) from a key person, in this case a project leader, and to observe the publicly available information provided by the community, such as website, forums, IRC channel and version control system. Also, the UKKOSS11 project material was used in the study as it included information related to the usability testing and inspection processes and practices. The case study report followed the master's thesis structure guidelines provided by the Department of Information Processing Science of the University of Oulu.

3.4 Collecting Evidence

Evidence collection is the fourth step of case study research. The four principles of data collection are: collect evidence from a number of different sources, assemble data into a database, maintain chain of evidence, and exercise care in using evidence from electronic sources such as social media. Also, triangulating evidence is important (Yin, 2014). The evidence for case study research can be collected for six sources: "documents, archival records, interviews, direct observation, participant-observation, and physical artifacts" (Yin, 2014, p. 102). One of the most important sources of evidence is interviews (Yin, 2014).

In other open source software studies, research material has been collected from publicly available online information provided by communities, such as version control systems (Mockus, Fielding & Herbsleb, 2002; Twidale and Nichols, 2005; Zanetti, 2012), forums (Zanetti, 2012), e-mail and mailing-lists (Mockus, Fielding & Herbsleb, 2002; Zanetti, 2012), bug tracking systems and IRC channel (Benson, Müller-Prove & Mzourek, 2004).

In this research the data was collected from the following material: e-mail correspondence with a project leader of the community including usability and open source software development -related questions (Appendix A & Appendix B), community's website, forum, and version control system, which in this case is GitHub. The community has also an IRC channel. The information provided by the community infrastructure followed direct observation method, but as forum discussion can be considered as social media, care should be exercised. As the UKKOSS11 project participated to the development, participant-observation was used to some extent. The gathered data was stored into a text document for further analysis.

The email correspondence included answers for usability and open source software development –related questions and discussion related to UKKOSS11 project including settling use cases for usability evaluation, a request for contributing more usability activities from the community and submitting early findings and complete reports of evaluations. In addition, all UKKOSS11 project material are included in the research material. The IRC channel was monitored for three months in 2013, but it did not have usability-related discussion. The version control system had an issue in which usability findings were addressed having a label "type:ux". In the community's website, search word "usability" gave no results and the website did not have a particular forum section for usability-related topics. However, by exploring the community's forum, a forum thread related to version 5.7 was found which included discussion about UKKOSS11 and usability of the system in general. Also, a project member's community profile page was used as he received community badges.

3.5 Analysis of Evidence

Analysis, the fifth step of case study research, is a not much developed aspect of the research method. One way to start the phase is to "play" and put information into different arrays, matrices or use other means to visualize the information and search for patterns, insights and concepts. In addition, there are four general strategies that can be used: relying on theoretical propositions, working data from bottom up, developing a case description and examining possible rival explanations. Also five analytical techniques exist which can be used with a general strategy: pattern matching, explanation building, time-series analysis, logic models and cross-case synthesis. (Yin, 2014.)

Relying on theoretical propositions means to be guided by the propositions that led to do the case study, i.e., the objectives and design which led to do a literature review and hypotheses or proportions. Working data from the "ground up" is a direct opposite of the first strategy, meaning that the strategy is data-driven and patterns can be found by playing with the data. Developing a case description is used if there are difficulties in using the first two strategies. In this strategy, data can be organized by topics and then a logical path or concepts can be found. The fourth strategy is to examine plausible rival explanations and it can be used alongside another strategy. Examples of rival explanation types are null hypothesis, threats to validity and investigator bias. (Yin, 2014.)

Alongside general strategies an analytical technique can be used. Pattern matching is one of the most desirable techniques in this research method. The technique means that there is a logical pattern in the evidence that can be predicted before collecting the evidence. If this happens, it increases the internal validity of the research (Yin, 2014). Another technique is explanation building which is very similar to pattern matching, but the procedure is more difficult and relevant to explanatory case studies. The aim of explanation building is to "analyze the case study data by building an explanation about the case", i.e., to explain why or how something happened (Yin, 2014, p. 147). Time-series analysis is the third technique. The technique is the same as time-series analysis in experiments and quasi-experiments, and has many patterns such as simple time series, complex time series and chronological sequences. The main point of time-series technique is to study *who* and *why* questions related to events over time. The fourth technique is logic models which examines more complex chains, e.g., cause-effect-cause-effect patterns meaning that a dependent variable becomes an independent variable for the next stage. The fifth and last technique is cross-case synthesis which is used only in studies with multiple cases. The technique treats individual case studies as separate studies and then the studies are synthetized (Yin, 2014).

In this study, the evidence was collected from the sources into a document-type database for further analysis and to search for patterns and insights related to the research objectives. The technique used in this research was pattern matching. As the first question compares the usability attitudes prior to and after the UKKOSS11, it involves time dimensions, and thus, time-series analysis was used. Time-series analysis was used by observing data related to the contributed usability activities based on the timetables of UKKOSS11 project, Concrete 5 versions and timestamps of forum posts. In pattern matching technique, the empirical observations were compared to the predicted patterns in the prior research literature when collecting the data and in further analysis. The theoretical propositions are summarized next.

The impacts to the community's attitude towards usability and usability activities can be observed through usability institutionalization process. First, it requires management's

support and commitment, after which a usability champion is selected who provides leadership, resources and coordination. Then, usability is fit into development process. Before hiring own usability specialists, organizations should first hire external usability consultants. Often, a wake-up call is required, which in the worst case scenario might be that the product turns out to be impractical and unusable. Next, usability should be sold to development and usability specialists or a team is hired. The specialist or the team carries out usability activities, train developers and evangelize usability in the organization (Schaffer, 2004).

In their literature review, Rajanen and Iivari (2013) found characteristic of HCI and OSS philosophies. The characteristics of the HCI philosophy are "usability specialists are user representatives, they act in informative, consultative, participative and designer roles, sticking with the user focus, knowing the user, speaking for and fighting for the user in the development [and] having decision-making power regarding important design solutions" (p. 2-3), and the characteristics of the OSS philosophy are "software as communal resource, voluntary collaboration, loosely coupled community interaction, different levels of involvement and status, gaining merit and reputation through contributing to the community, especially high quality code, scratching your own itch [and] talk is cheap, show me the code" (p. 3). The different outcomes of HCI philosophy in two case studies were: "usability specialists as user representatives, in consultative and participative roles, sticking with the user focus" and in "designer roles" (p. 5). And the findings OSS philosophy were: "community interaction, gaining merit and reputation, representing end users, acting as bug reporters and bug fixers, interacting with core developers" and "acting as committers, adopting developer focus and developer and gamer roles" (p. 5). As the system in this research is not a game, adopting gamer roles can be omitted from the possible findings.

3.6 Sharing the Results

Sharing the results is the last step of case study research process. The purpose of sharing is to let other people know about the results and findings. First the audience must be defined whether the sharing occurs in written or spoken form. It is recommended to show necessary evidence for readers so that they can reach their own conclusions. The sharing process should start early and it can include textual and visual materials (Yin, 2014). There are six different compositional structures for reporting case studies: linear-analytic, comparative, chronological, theory-building, suspense, and unsequenced structures" (Yin, 2014, p. 176). For explanatory studies, all the previously mentioned structures can be used except for unsequenced structure. The sharing process ends with an iterative review and re-compose cycle. (Yin, 2014).

The main audience of this thesis is people with academic background, for example, the thesis supervisor, opponent and students plus the open source community. The primary goal of this research is to produce a master's thesis, and thus, the sharing will occur in written form following the thesis instructions provided by the Department of Information Processing Science at the University of Oulu. The results of HCI and OSS philosophies can be summarized and presented in tables similarly to the journal article by Rajanen and Iivari (2013).

4. Findings

This section reports the collected and analysed findings according to case study research methodology (Yin, 2014). First, the impacts on the community's usability attitudes are presented, and then the existed characteristics of HCI and OSS philosophies.

4.1 Impacts on the Community

This chapter gathers the findings related to the impacts of UKKOSS11 project to the target open source software project. First, the findings based on questions sent to a project leader are analysed and compared, and the other observations gathered mainly from the community's website and email correspondence with the project leader.

Based on the questions sent to a project leader of the community in October 2013, no external usability activities had been conducted for the OSS project before UKKOSS11 project. He also mentioned that the community did not have already identified usability issues, although some had been mentioned through anecdotes and their own experiences. In addition, it was asked whether the community has a To Do list for usability experts and do the community have a person on charge for the usability-related issues, i.e., a usability champion. The answers were that a To Do list does not exist, but it would be helpful to have one, and that the community does not have a dedicated usability champion, but the person in responsible for usability would be the project leader himself or a person who is in lead of documentation and training.

In April 2016, another set of questions were sent to the project leader. The project leader reported that a couple of usability studies had been done by third parties who tested also their own add-ons. The third party studies were not commissioned by the OSS community. Moreover, it was mentioned that the community had *"*gotten pretty far without [formal usability tests] and just using our own experiences with clients and with using our own software (and commissioning informal responses from our community) but [UKKOSS11] so far as has proven very helpful". When asked about already identified usability issues, the project leader answered that the "community might say we do". However, the project leader mentions that he is not aware of other issues than found in UKKOSS11 project. In addition, he explains that many of the issues "are the result of incomplete and insufficient documentation […] of how to use concrete5" and its features. Furthermore, the project leader answered that they still do not have a To Do list, but it "sounds like a good idea" and the situation regarding to usability champion had not changed: there was no dedicated usability champion, but the project leader added that the responsibility would probably fall to him as he is responsible for the back-end code and functionality as well as for the usability and other objectives for the system. He underlines that he is concerned that the systems "is sufficiently easy enough for all the different types of users of concrete5". Compared to the questions sent in 2013, three new questions were added in 2016 to investigate the perceived impacts on short-term and long-term effects of UKKOSS11 and how beneficial UKKOSS11 project as a whole was. The project leader mentioned that UKKOSS11 project had short-term effects on the community as it revealed some usability issues that they were not addressing. The project leader mentioned that it is too soon to tell about the long-terms effects of the usability project because there has not been time to address the

specifics of the usability findings. When asking how beneficial the usability project was using Likert scale from 1 to 5 (1 = not beneficial, 5 very beneficial), the project leader gave rating 4. Moreover, he alwasys showed gratitude after sending reports of usability activities.

When comparing the answers given in 2013 and 2016, there was no change in setting up a To Do list for usability experts and a usability champion was no named, although the project leader could identify himself as one of the potential candidates for the role. There was a change in contributed usability activities as some third parties had contributed usability issues as a side product of testing their own add-ons.

During the first phase, a How To document was written for the community on conducting usability tests and inspections for the software including the basics of each method and whom to contact for getting use cases as an input for the usability activities. Concrete5's Tutorial section in the website includes many How To tutorials including mostly technical topics aimed at web developers. However, the document was never published.

The first phase of UKKOSS11 project found usability issues in the text-editor called TinyMCE used in the system. The community changed the text editor to Redactor in the work in progress (WIP) version of 5.7. After changing the text editor and releasing the WIP version of the software, there was a forum page for voting for the text editor used in the official 5.7. release. When asking the project leader whether the reason for the change was the found usability issues, he mentioned that it was one of the reasons. Also, the core team had observed that creating a page in version 5.6. was difficult. As the page could be created through "Create sub-page" functionality, the usability tests showed that test participants created the pages to the wrong location as they were not fully aware of their location in the site hierarchy when starting the activity and thus, had to move the page to the correct location. In version 5.7. the activity was changed to "Create page" which asked the target location before the page was saved.

After the first phase of the usability project, the project leader asked the UKKOSS11 members to "consider doing a compare and contrast between the old version of concrete5 and the new [version]". He also mentioned that some of the findings from the first phase of UKKOSS11 project would be incorporated to the new release.

Nonetheless, some usability-related issues exist in the version control system that is publicly available on GitHub. For example, the usability issues from the third phase of UKKOSS11 are located in the GitHub, and the project leader mentions that, in general, usability issues found by external usability contributors will be fixed by the core team or they hope that the community will help to fix the issues. A label named "type: ux" can be found on Concrete5's GitHub issue tracker (GitHub).

Even though the community does not have a particular forum for usability discussion, discussion about the subject can be still found. For example, in a forum discussion titled "5.7 discussion" there is discussion about the change of the text-editor, usability-related conversation in which also UKKOSS11 project was mentioned. The project leaders wanted the product to be user-friendly for the whole user base, but some members thought that web developers would hate that kind of transformation as they changed to Concrete5 from other CMSs that were not good for developers in the first place. A project leader mentioned that a good measure for the software to be easy to use for "Average Joes and Janes" and a question was raised that would the project leader's mom accomplish a user task. The project leader also mentioned that they had analysed the user interfaces of other CMSs to see what could be done better in Concrete5. The user interfaces of the software were almost completely renewed in version 5.7., and

people expressed their opinions and experiences regarding to the changes. One of the project leaders mentioned that a usability study was conducted, referring to UKKOSS11 project, for the earlier version of the software, and the findings were submitted to another project leader. He mentions that one of the findings was that creating a page was counter-intuitive for first-time users. Two of the community members criticized the contributed usability activities. The first member wrote:

> *[A]gain…I'd say that's the wrong audience.*
>
> *[Y]ou're effectively basing a major paradigm shift on a bunch of college 20 year olds?*
>
> *Are you more experienced and knowledgeable now, or when you were 20? lol*
>
> *I'm telling you, I have never, in training hundreds of people EVER had anyone complain about creating pages or adding blocks. Ever.* (Concrete5.)

Another member criticized that:

> *Are CORE team and hard core [developers] best UX guinea pigs - apart from Fin[n]ish 20yr old students of course? Some of the best feedback can be had from sitting people down, videoing them and watching them 'muddle through' the tasks (see Steve Krug[']s "don[']t me think"). If we start with "add a page", "visit a different page" , then "go back and edit that page you made" I am pretty sure we'd get some interesting data to throw into the mix.* (Concrete5.)

After all the criticism, the project leader corrected that:

> *No, a usability study isn't two 20 year olds just offering their opinions as fact. While that may fly here, what they did was sit a cross section of normal people down and ask them to accomplish real tasks. They found a number of things, most of which confirmed our own experiences with training clients.* (Concrete5.)

4.2 Existence of HCI Philosophy

In the beginning of the usability project, two main user groups were identified: content managers with no technical background and web developers. The tested user tasks were settled with a project leader so that the most common and basic user tasks were covered in tests and inspections. After defining the user profiles, test participants were recruited and selected based on the user profile criteria. Test participant from both user groups were recruited.

The usability test sessions were recorded so that the performance and behaviour of the test participants can be analysed further. Improvement suggestions for the interfaces were made based on the usability findings found in test session observations and the post-session analysis. After the test sessions, the participants were asked questions such as, "What was good in the system?", "What was bad in the system?" and "How would you improve the system?". The comments were included in the usability test reports which were reported further to the project leader.

Cognitive walkthrough aims to evaluate the ease of use of a product from the point of view of users with no user training. Cognitive walkthroughs were conducted in the first and second parts of the usability project.

Even though there was usability-related discussion in the community forum, the usability project members did not participate to the discussion. One reason for this was that the discussion was noticed two weeks after the most active discussion. Thus, it was decided not to participate to the discussion so that the researchers would not interrupt the discussion and let the community members and leaders discuss on their own.

Moreover, the usability tests and inspections were conducted for functional user interfaces rather working with the developers in early phases in the development when usability tests and inspection could be conducted for paper prototypes.

Some of the usability-related questions, such as To Do -list for usability specialists and is there a usability champion, asked from the project leader included issues related to management level usability and can be considered as suggesting consultative ideas.

4.3 Existence of OSS Philosophy

Before the actual usability project started, one of the UKKOSS11 project team member proposed Concrete5 to be the target community. He was already a passive user of the software and a community member. Another reason for selecting the community was that he had used the software to develop a website for his customer, and he wanted to know how usable the software is for first-time users with less technical skills. The aim was to know what kind of training is required when delivering a website with the software so that end users could work on their own after delivery.

In this kind of a project that is related to the studies, collaboration is not as voluntary as in traditional collaboration between OSS users or developers and the core team of the community. However, the second project, usability inspections for version 5.7.WIP, was more voluntary as the course credits provided did not belong to the mandatory studies of the curriculum. The third phase consisted of five members of which four were students participating to usability testing course. The fifth member, the author of this thesis, participated to the project to obtain more research data for this thesis.

During the UKKOSS11 project, communication with the community occurred only via a project leader. Also IRC channel was monitored, but there was not UKKOSS11 or usability-related discussion. During the periods between the project phases there was no correspondence with the community.

The author of this thesis was a community member and a passive user before UKKOSS 11 project started. The goal of UKKOSS11 project was to evaluate the usability of the product by using different usability testing and inspection methods. By evaluating the usability of the system and submitting usability-related issues, the project members became bug reporters. Also written suggestions to fix the defects were sent, which can be interpreted that the members were also designers to some extent.

After reporting the findings from the first phase of the project, the contact person of UKKOSS11 project received two badges from the core team. Also, after facing a problem in which form submissions were sent to spam box and thus left unnoticed first, a project member asked help from the project leader who assisted with the problem. The project member saw a forum post regarding the same problem and gave assistance, which resulted to receiving another badge. The badges received from usability

contributions were “Silver Star I” and “Silver Star II” and their explanations are respectively “accomplished something cool” and “accomplished a few cool things” (Concrete5). The badge received from helping in the forum was “First Reply” which is described as “Took the time to not only start a new thread, but also post to someone else’s” (Concrete5). This proves that gaining merit can be achieved by other means than only contributing high quality code. In the project, we did not contribute code, but instead, suggestions to fix the found usability-related defects and a How To document, which has not been published up to this date.

At any point of the project, code patches were not requested or sent. According to the email correspondence, the project leader was grateful for submitting the results and hoped that community members would fix the usability-related bugs mentioned in GitHub’s issue tracker.

5. Discussion and Implications

From the beginning of the usability project, the OSS project leader indicated that the usability project is welcome and gave support by helping to identify test tasks. After the first phase of the project, when asked whether there is a usability champion in the project, the project leader said that there is no formal usability champion, but named himself as the primary person having the responsibility. After the first phase, he asked whether a new round of tests can be conducted to compare the two versions. However, when asking later whether the community has a usability champion, a To Do list for usability specialists or whether they had conducted usability tests, no change had occurred between October 2013 and April 2016. Even though the project leader identified himself as a possible usability champion and acted as one by requesting external usability activities from UKKOSS11 team, addressing reported usability issues to issue tracking, and by expressing the need for usability in the community forum, no official usability champion nomination was done.

As the UKKOSS11 project communicated the usability activities only with the project leader, we can assume that members outside the core team did not know the UKKOSS11-related details about test activities such as the used usability test methods, test tasks and test participants. In a version 5.7. forum discussion, there was usability-related discussion and some of the community members criticized the contributed usability activities by stating that the opinions of 20-year-old college students might not be trustworthy, and one member suggested that the core team ought to conduct usability tests by observing how test participants perform referring to usability test method. One of the project leaders mentioned that this was exactly what the UKKOSS11 project did, after which the criticism towards the UKKOSS11 project ended.

Also the software was modified according to the found usability-issues. For example, when a user logs into the system for the first-time, in version 5.7.5.6., there is a element which lists links to How To documentation for common activities. A new field was implemented to form functionality's settings so that users can freely define *Submit* button's text. Also a text editor was changed and one the reasons for that was the found usability-related issues in the previous one.

In conclusion, from the core team's point of view, the UKKOSS11 was seen as beneficial and helpful. Some sort of wake-up call did occur as the project leader asked for more usability activities after the first phase of the UKKOSS11 project referring to "hiring" external usability consultants and that usability activities were fit into the development process. In addition, some of the findings were implemented. Also other community members cared about the usability of both main user groups, and some members knew to some extent how to test usability. Based on the aforementioned facts, and that a usability champion was not named and internal usability experts were not hired, it can be concluded that complete institutionalization of usability did not occur.

Main user groups, web developers and content managers, were identified and the product was tested using test participants recruited based on the user profiles. Also, in general, cognitive walkthrough inspection method uses first-time users' point of view. The feedback from users and results from usability tests and inspections including suggestions to fix the issues were submitted to the project leader. The aim was mainly

to stick with the user focus, but creating suggestions can be also interpreted as designing activity. The questions asked from the project leader included some consultative issues such as naming a usability champion and creating a To Do list for usability specialists. These findings imply that usability specialists were user representatives in informative, consultative and designer roles, knowing the user, speaking for the user in the development and sticking with the user focus.

Based on the findings which were reported by the project leader regarding to changing the text-editor used in the system and changing the create page activity showed that the contributed usability test results had some impact on the community, but no actual decision-making power.

Before the target community for the usability project was selected, one of the project members who was already a community member of the OSS project, suggested Concrete5 as the target community because he wanted to know how well first-time users with no technical background would perform in the core use cases such as creating a page. During the project his role transformed from passive user to bug reporter. The second part of the project was done voluntarily because the course credits provided by the university did not belong to the mandatory courses. Also helping another community member to solve a problem in the forum was voluntary activity and loosely coupled community interaction. Both contributing usability and helping another member were rewarded with badges. Through the criticism, there were changes in the reputation: first, the reputation of usability could have been seen as negative, but as one of the project leaders corrected the statements, it possibly changed the reputation to a more positive light. Altogether, these findings imply that the following characteristics of OSS philosophy co-existed: software as communal resource, voluntary collaboration, loosely coupled community interaction, acting as bug reporters and designers, gaining merit and reputation through contributing to the community and scratching your own itch.

Table 1. Existence of HCI and OSS philosophies during UKKOSS11 project.

	HCI Philosophy	**OSS Philosophy**
Observations	Usability specialists were user representatives in informative and consultative roles, knowing the user, speaking for the user in the development and sticking with the user focus	Software as communal resource, voluntary collaboration, loosely coupled community interaction, acting as bug reporters and designers, gaining merit and reputation through contributing to the community, scratching your own itch

The existence of HCI and OSS philosophies in this study is very similar to the prior study but has a few differences as also did the prior two cases. This study did not have acting in participative and bug committers roles, adopting developer focus and developer and gamer roles. Also, there was no decision-making power regarding to important design solutions. The last difference, adopting a game roles, is obvious as the target system was not a game project. The instantiation of OSS characteristic "scratching your own itch" was found which was not found in the prior research.

In the prior literature, it is mentioned that gaining merit by contributing usability activities is rare (Rajanen & Iivari, 2013), but this project succeed in getting community badges. As suggested in the prior literature (Rajanen, Iivari and Anttila, 2011) participative approach is more likely to yield successful results in the target community compared to consultative approach. The usability project could have been successful by submitting code patches, designing mock-ups, participating more to user interface design and sharing knowledge about usability testing and inspection practices. In addition, the usability project could have helped to establish a usability discussion infrastructure (Nichols & Twidale, 2003) including feedback on mock-ups, prototypes, and custom builds (Terry, Kay & Lafreniere, 2010), human level on top of technical level (Hedberg & Iivari, 2009), establishing a user interface specification document or writing interface guidelines and user documentation (Benson, Müller-Prove & Mzourek, 2004).

6. Conclusions

This study examined usability evaluation project's impacts to an open source software community and the existence of core human computer interaction and open source software philosophies.

The core team's attitude towards contributed usability activities were welcoming and supportive, and according to the software project's objectives, forum discussion and email correspondence, they seemed to care about usability, especially when considering users with no technical background. Without knowing the usability evaluation methods used in the usability project, some of the community members criticized the contributed usability project in the community's forum by assuming that it included only opinions of 20-year-old college students, but one of the influential project leader corrected that usability testing involving test participants was conducted, which silenced the criticism. The community had not conducted any usability activities on their own, and felt that they had gone far without them, but after the first usability contributions they requested external consultancy in terms of another study which indicates that a wake-up call to some extent had occurred. However, no signs of instituting usability to the organization was found.

The found characteristics of core HCI philosophy in this study are that usability specialists were user representatives in information and consultative roles, knowing the user, speaking for the user in the development and sticking with the user focus. The existence of core OSS philosophy characteristics are that software was considered as communal resource, collaboration was voluntary, interaction with the community was loosely coupled, usability experts acted as bug reported and designers to some extent gaining merit and reputation through contributing to the community. The idea to select Concrete5 as the target group was that one project group member was interested how first-time users could use the systems without help, which refers that there was an itch which the project member wanted to scratch.

This research contributes to the literature by adding a case study of introducing usability activities into open source software development examining the existence of HCI and OSS philosophies. Compared to the prior HCI and OSS philosophy study, a new characteristic in OSS philosophy, "scratching your own itch", was found.

One of the limitations for the study is that it was conducted by students rather than usability professionals, even though the students had completed courses related to human computer interaction. Another limitation is that because the primary motivation for the students was to complete courses, there is a tendency to complete the required assignments related to the course without doing anything extra that would further help to institutionalize usability in the community, for example, by setting up usability-related discussion areas, close interaction with user interaction designers or setting up workshops related to usability. However, there are cases when students have gained merits and succeeded in introducing usability activities to OSS context.

Another limitation to the study is that communities, relationships, decisions and projects are less concrete units of analysis than individuals, smalls groups, organizations and partnerships (Yin, 2014). During UKKOSS11, the usability project team changed between the second and third phase as the other member from first two phases was

replaced by four students participating to usability testing course. In addition, as open source software communities are found to be loosely coupled relying on voluntary collaboration, also changes in the community might have occurred.

All of the discussion in the community website, IRC channel and GitHub issue tracker was not read and as the physical distance between the core team was huge, it was not possible to collect and analyze everything that could be considered as research material in the given time frame and without funding. The lack of research material could have further affected the conclusions of this study.

Also, another limitation is that the use of social media is not recommended by research method literature (Yin, 2014). In this thesis, forum discussion is used, but it did not have much effect on the conclusions. Moreover, the analysis of the research data and the derived conclusions are dependable on the researcher's interpretation which establishes another limitation to the study.

More research should be done to understand how HCI and OSS philosophies are related to create a wake-up and further leading to the institutionalization of usability in open source development projects. In addition, more data is needed to know what are the best approaches to introduce usability into OSS development by varying factors such as leadership models, structures, project size, development method and domain.

References

Aberdour, M. (2007). Achieving quality in open-source software. *Software, IEEE*, 24(1), 58–64. doi: 10.1109/MS.2007.2

Androutsellis-Theotokis, S., Spinellis, D., Kechagia, M., & Gousios, G. (2010). Open Source Software: A Survey from 10,000 Feet. *Foundations and Trends® in Technology, Information and Operations Management, 4(3-4)*, 187–347. doi:10.1561/0200000026

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)*, 303-312

Arhippainen, L. (2009). *Studying User Experience: Issues and Problems of Mobile Services*. Acta Universitatis Ouluensis, A Scientiae Rerum Naturalium 528

Benson, C., Müller-Prove, M., & Mzourek, J. (2004, April). Professional Usability in Open Source Projects: GNOME, OpenOffice.org, NetBeans. *Proceeding CHI'04 Extended Abstracts on Human Factors in Computing Systems*, Vienna, Austria (pp. 1083-1084).

Bevan, N. (2000). *Cost Benefit Analysis version 1.1. Trial Usability Maturity Process*. Serco Usability Services

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

Bjerknes, G., & Bratteteig, T. (1995). User Participation and Democracy: A Discussion of Scandinavian Research on System Development. *Scandinavian Journal of Information Systems, 7(1)*, 73-98

Boldyreff, C., Lavery, J., Nutter, D., & Rank, S. (2003). Open-Source Development Processes and Tools. Proceedings of the 3rd Workshop on Open Source Software Engineering, International Conference on Software Engineering (ICSE'03), pp. 15-18.

Bonaccorsi, A., & Rossi, C. (2003). Why Open Source Software Can Succeed. *Research Policy, 32(7)*, 1243–1258. doi:10.1016/S0048-7333(03)00051-9

Boivie, I., Gulliksen, J., & Göransson, B. (2006). The lonesome cowboy: A study of the usability designer role in systems development. *Interacting with Computers, 18(4)*, 601–634. doi:10.1016/j.intcom.2005.10.003

Çetin, G., & Göktürk, M. (2008). A Measurement Based Framework for Assessment of Usability-Centricness of Open Source Software Projects. 2008 In *Signal Image Technology and Internet Based Systems, 2008. SITIS. IEEE International Conference on* (pp. 585-592). IEEE

Chattratichart, J., & Lindgaard, G. (2008). *Proceeding of the twenty-sixth annual CHI conference Extended Abstracts on Human factors in computing systems - CHI '08, Florence, Italy, April 5-10.* New York, USA: ACM

Clemmensen, T. (2004). Four approaches to user modelling — a qualitative research interview study of HCI professionals' practice, *Interacting with Computers*, *16(4)*, 799–829. http://doi.org/10.1016/j.intcom.2004.04.009

Concrete5. http://www.concrete5.org

Davis, F. D. (1989), Perceived usefulness, perceived ease of use, and user acceptance of information technology, *MIS Quarterly*, 13(3): 319–340

DeLone, E., & McLean, W. (2003). The DeLone and McLean Model of Information Systems Success: a Ten-Year Update. *Journal of Management Information Systems*, 19(4), 9-30.

Donahue, G. (2001). Usability and the Bottom Line. *IEEE Software 18(1)*, 31–37

Dumas, J.S., & Redish, J.C. *(1999). A Practical Guide to Usability Testing* (Revised ed.). Portland, USA: Intellect

Ehrlich, K., & Rohn, J. (1994). Cost Justification of Usability Engineering: A Vendor's Perspective. In: Bias, R., Mayhew, D. (eds.) *Cost-Justifying Usability*, pp. 73–110. Academic Press, London

Fitzgerald, B. (2011). Open Source Software: Lessons from and for Software Engineering. *IEEE Computer*, (October), 25–30.

Fogel, K. (2013). *Producing Open Source Software – How to Run a Successful Free Software Project*. O'Reilly Media

GitHub. Concrete5 repository. https://github.com/concrete5/concrete5

Hall, J. (2014). Usability Themes in Open Source Software. *Scientific and Technical Communication (STC) Plan C Research Paper*

Hedberg, H., & Iivari, N. (2009). Integrating HCI Specialists into Open Source Software Development Project. *Proceedings of the 5th IFIP WG 2.13 International Conference on Open Source Systems*, OSS 2009, (251-263) Skövde, Sweden

Hedberg, H., Iivari, N., Rajanen, M., & Harjumaa, L. (2007, May). Assuring Quality and Usability in Open Source Software Development. In *Emerging Trends in FLOSS Research and Development, 2007. FLOSS'07. First International Workshop on* (pp. 2-2). IEEE

Hevner, A., March, S., Park, J., & Ram, S. (2004). Design Science in Information Systems Eesearch. *MIS Quarterly*, 28(1), 75–105

Holzinger, A. (2005). Usability Engineering Methods for Software Developers. *Communications of the ACM*, 48(1), 71–74. doi:10.1145/1039539.1039541

Iivari, N. (2006). Understanding the work of an HCI practitioner. *Proceedings of the 4th Nordic Conference on Human-Computer Interaction Changing Roles - NordiCHI '06*, (October), pp. 185–194. http://doi.org/10.1145/1182475.1182495

ISO 13407 (1999). Human-centred design processes for interactive systems. International Organization for Standardization

ISO 18529 (2000). Human-centred lifecycle process descriptions. International Organization for Standardization

ISO 9241-11 (1998). Ergonomic requirements for office work with visual display terminals (VDTs). International Organization for Standardization, 45

ISO 9126-1 (2001): Software Engineering Product Quality Part 1: Quality Model. Internation Organization for Standardization, 27

Karat, C.-M. (1994). A Business Case Approach to Usability Cost Justification. In: Bias, R., Mayhew, D. (eds.) *Cost-Justifying Usability*, pp. 45–70. Academic Press, London

Kishida, K., & Ye, Y. (2003). Toward an Understanding of the Motivation of Open Source Software Developers. *Proc. 25th Int'l Conf. Software Eng. (ICSE 03), IEEE CS* Press, 419–429.

Krug, S. (2010). *Rocket Surgery Made Easy – The Do-It-Yourself Guide to Finding and Fixing Usability Problems*. Berkeley, New Riders.

Lakhani, K. R., & Wolf, R. (2005). Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, In J. Feller, B. Fitzgerald, S. Hissam, and K. R. Lakhani (Eds.), *Perspectives on Free and Open Source Software.* MIT Press: Cambridge, Massachusetts

Lanamäki, A., Rajanen, M., Öörni, A., & Iivari, N. (2015) Once You Step Over the First Line, You Become Sensitized to the Next: Towards a Gateway Theory of Online Participation. In the Proceedings of the 36th International Conference on Information Systems (ICIS2015). Fort Worth, USA.

Ljungberg, J. (2000). Open Source Movement as a Model for Organising. *European Journal of Information Systems, 9(4)*, 208-216.

Mayhew, D., Mantei, M. (1994). A Basic Framework for Cost-Justifying Usability Engineering. In Bias, R.G. & Mayhew, D.J. (Eds.), *Cost-Justifying Usability,* Academic Press: London

Marcus, A. (2005). User Interface Design's Return on Investment: Examples and Statistics. In Bias, R.G. & Mayhew, D.J. (Eds.), *Cost-Justifying Usability*. San Franciso, CA: Elsevier, Inc.

Marghescu, D. (2009). Usability Evaluation of Information Systems: A Review of Five International Standards. In Information Systems Development (pp. 131-142). Springer: US.

Mockus, A., Fielding, R., & Herbsleb, J. (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Eng. and Methodology (TOSEM)*, 11(3), 309–346.

MOT Oxford Dictionary of English.

Muller, M., Matheson, L., Page, C., & Gallup, R. (1998). Methods & tools: participatory heuristic evaluation. *ACM interactions*, 5(5), 13–18.

Nadeau, T. (1999). *Learning from Linux – OS/2 and the Halloween Memos*. Retrieved from http://www.os2hq.com/archives/linmemo1.htm

Nakakoji, K., Yarnamoto, K., Nishinaka, Y., Kishida, K., & Ye, Y. (2002). Evolution Patterns of Open-Source Software Systems and Communities, *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, 76-85

Nichols, D., & Twidale, M. (2003). The Usability of Open Source Software. *First Monday, 8(1)*.

Nichols, D., & Twidale, M. (2006). Usability Processes in Open Source Projects. *Software Process: Improvement and Practice, 11(2)*, 149-162

Nielsen, Jakob. (1993). *Usability engineering*. Boston: Academic Press, cop.

Nielsen, J. (1994). Heuristic Evaluation. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods*. New York, USA: John Wiley & Sons.

Oezbek, C., Prechelt, L., & Thiel, F. (2010, May). The onion has cancer: Some social network analysis visualizations of open source project communication. *In Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development,* 5-10. ACM.

Oinas-Kukkonen, H., & Oinas-Kukkonen, H. (2013). *Humanizing the Web: Change and Social Innovation*. New York: Palgrave and Macmillan.

Open Source Initiative. The Open Source Definition. Retrieved from http://opensource.org/docs/osd

Pinelle, D., Wong, N., & Stach, T. (2008). Heuristic Evaluation for Games: Usability Principles for Video Game Design. C*HI'08 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*

Rajanen, M., & Iivari, N. (2007). *Usability Cost-Benefit Analysis: How Usability Became a Curse Word?* Proceedings of 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007. doi: 10.1007/978-3-540-74800-7_47

Rajanen, M., & Iivari, N. (2010). *Traditional Usability Costs and Benefits – Fitting them into Open Source Software Development*. In proceedings of the 18th European Conference on Information Systems (ECIS). Pretoria, South Africa.

Rajanen, M., & Iivari, N. (2013). *Open Source and Human Computer Interaction Philosophies in Open Source Projects – Incompatible or Co-Existent?* Proceedings of International Conference on Making Sense of Converging Media.

Rajanen, M., Iivari, N. (2015a). *Examining Usability Work and Culture in OSS*. In the Proceedings of the 11th international Conference on Open Source Systems (OSS 2015). Florence, Italy.

Rajanen, M., Iivari, N. (2015b). *Power, Empowerment and Open Source Usability*. Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. Seoul, Korea.

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). *Introducing Usability Activities into Open Source Software Development Projects - a Participative Approach*. NordiCHI '12, Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design, 683-692.

Rajanen, M., Iivari, N., & Lanamäki, A. (2015). *Non-Response, Social Exclusion, and False Acceptance: Gatekeeping Tactics and Usability Work in Free-Libre Open Source Software Development*. In the Proceedings of 15th IFIP TC.13 International Conference on Human-Computer Interaction (INTERACT 2015). Bamberg, Germany

Raza, A., Capretz, L., & Ahmed, F. (2011). Users' Perception of Open Source Usability: An Empirical Study. *Engineering with Computers, 28(2)*, 109-121.

Raza, A., Capretz, L., & Ahmed, F. (2012). Usability Bugs in Open-Source Software and Online Forums. *IET Software, 6(3),* 226-230

Raymond, E. (1999). *The Cathedral and the Bazaar*. Knowledge, Technology & Policy, 1–40. Retrieved from http://link.springer.com/article/10.1007/s12130-999-1026-0

Rolandsson, B., Berquist, M. & Ljungberg, J. (2009). *Open Source Programmers' Strategies to Cope with Ideological Tensions.* In the Proceedings of International Conference on Organizational Learning, Knowledge and Capabilities, April 26-28, Amsterdam, The Netherlands

Rosen, L. (2004). *Open Source Licensing - Software Freedom and Intellectual Property Law*. Retrieved from http://www.rosenlaw.com/oslbook.htm.

Rubin, Jeffrey. (2008). *Handbook of usability testing: how to plan, design, and conduct effective tests.* Indianapolis, IN: Wiley, cop.

Schaffer, E. (2004). *Institutionalization of Usability: A Step-by-Step Guide.* Boston: Addison-Wesley

Schramm, W. (1971). *Notes on Cases Studies of Instructional Media Projects.* Working paper for the Academy of Educational Development, Washington, DC

Shah, S. K. (2006). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science, 52(7)*, 1000–1014. doi:10.1287/mnsc.1060.0553

Sharp, H., Rogers, Y., & Preece, J. (2007). *Interaction Design: Beyond human-computer interaction*. 2nd ed. John Wiley & Sons.

Spyridakis, J. K., (1992). Conducting Research in Technical Communication: The Application of True Experimental Designs, *Technical Communication*, Fourth Quarter, 1992, pp. 607-624

Taylor, S., & Todd, P. A. (1995). Understanding Information Technology Usage: A Test of Competing Models. *Information Systems Research, 6(2)*, 144-176.

Terry, M., Kay, M., & Lafreniere, B. (2010). Perceptions and Practices of Usability in the Free/Open Source Software (FOSS) Community. *Proceedings of the 28th International Conference on Human Factors in Computing Systems - CHI '10.* Atlanta, GA, USA

Twidale, M., & Nichols, D. (2005). Exploring Usability Discussions in Open Source Development. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences – HICSS '05*.

Ye, Y., & Fischer, G. (2007). *Designing for Participation in Socio-technical Software System*. In Stephanis, C. (Ed.), Proceedings of 4th International Conference on Universal Access in Human-Computer Interaction, Beijing, China, pp. 312-321. Springer, Heidelberg.

Ye, Y., & Kishida, K. (2003). *Toward an understanding of the motivation of open source software developers*. Proceedings of the 25th International Conference on Software Engineering

Yin, R. (2014). *Case Study Research: Design and Methods – Fifth edition*. USA: SAGE Publications.

Venkatesh, V., & Davis, F. D. (2000). A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies. *Management Science, 46(2)*, 186–204. doi:10.1287/mnsc.46.2.186.11926

Venkatesh, V., Morris, M., Davis, G., & Davis, F. (2003). User acceptance of information technology: Toward a unified view. *MIS quarterly*, *27(3)*, 425–478.

Venkatesh, V., Thong, J. Y. L., & Xu, X. (2012). Consumer Acceptance and Use of Information Technology: Extending the Unified Theory of Acceptance and Use of Technology. *MIS Quarterly*, *36*(1), 157–178.

von Hippel, E. (2007). Horizontal innovation networks - by and for users, *Industrial and Corporate Change, 16(2)*, 1–23

Walsham, G. (1995). Interpretive case studies in IS research: nature and method. *European Journal of Information Systems*, *4*(2), 74–81.

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems*, *15*(3), 320–330.

Wania, C., Atwood, M., & McCain, K. (2006). How do Design and Evaluation Interrelate in HCI research? P*roceedings of the 6th ACM conference on Designing Interactive systems - DIS* '06, 90-98.

Wharton, C., Rieman, J., Lewis, C,. & Polson, P. (1994). *The Cognitive Walkthrough Method: A Pracititioner's Guide*. In J. Nielsen & R. Mack (Eds.), Usability Inspection Methods. New York, USA: John Wiley & Sons.

Wilson, C., & Coyne, K. P. (2001). The whiteboard: Tracking usability issues: to bug or not to bug? *Interactions, 8*, 15–19. doi:10.1145/369825.369828

Appendix A. Questionnaire 1

1) Have you had earlier previous usability activities in the Concrete5 project?

2) Do you have a ToDo-list for usability experts so that they can take tasks and contribute to the project?

3) Do you have a person who is in charge for usability related issues?

4) What are the main communication methods in the project (IRC, forums, mailing lists, etc.)?

5) What are the hierarchies within the community (top core developers, strict hierarchy, meritocratic hierarchy, loose hierarchy, democratic hierarchy, etc.)?

6) Do you have already identified usability issues (before us)?

Appendix B. Questionnaire 2

1) Have you had other than our usability activities in the Concrete5 project?

2) Do you have a ToDo-list for usability experts so that they can take tasks and contribute to the project?

3) Do you have a person who is in charge for usability-related issues (so-called usability champion)?

4) What are the main communication methods in the project (IRC, forums, mailing lists, etc.)?

5) What are the hierarchies within the community (top core developers, strict hierarchy, meritocratic hierarchy, loose hierarchy, democratic hierarchy, etc.)?

6) Other than ours, do you have identified usability issues?

7) Did our usability activities have any short-term effects on the community?

8) Did our usability activities have any long-term effects on the community?

9) Rate from 1 to 5 how beneficial our usability activities were (1 = not beneficial, 5 = very beneficial)?