Finding Out What Might Cause Poor Usability in Open Source Software Products

Matias Ylipelto, Henna Nissinen & Eero Parviainen

Abstract

As the popularity of *open source software* (OSS) increases, the amount of novice users not participating in the development process also increases and the usability of the software becomes more and more important issue. O*pen source software development* (OSSD) is typically decentralized and engineer-driven, meaning that design decisions usually are not made from top-down perspective; OSSD projects can come into being simply because certain single developer wants to develop something for him/herself. The usability is not always good in OSS and it can very easily scare potential users away from the products, which might otherwise serve their purpose well. We conducted a literature review with the goal of finding out what might cause poor usability in OSS products. We found out that low priority of usability, lack of testing, narrow target audiences, small project size, consensus reaching difficulties, and lack of consistency were all contributing to bad usability in OSS products. By examining the results we composed a set of guidelines which we think can be useful for any OSSD project if they want to invest to improve their product's usability.

1. Introduction

The purpose of this study is to find out the reasons behind different usability problems in OSS products. OSSD differs from traditional software development in its approach and decision making. We aim to pay attention to these OSSD characteristics as we study the factors that contribute to bad usability. Our research method is literature review.

Usability is not good in some of the open source software products and the worst case scenario is that bad usability can drive the users completely away from the products. Users may feel that software is inefficient if the user interface is not good and it is too difficult to use, even though it could get the job done. The motivation for our study is that by recognizing the reasons behind usability problems we can possibly help OSSD communities achieve better understanding of the whole picture when it comes to the usability of their products.

The reasons for usability problems in open source projects have been studied by multiple authors. In general, OSS developers tend to develop software for their own use, from their own point of view, without considering the fact that nowadays more and more novice users are using open source software (Nichols & Twindale, 2005; Pemberton, 2004; Raza, Capretz, & Ahmed, 2010; Raza, Capretz, Ahmed, 2012). It is suggested that the reason for OSS developers not being interested in improving usability is the lack of challenges and incentives in usability related tasks (Andreasen, Nielsen, Schrøder & Stage, 2006; Nichols & Twindale, 2005). One way of explaining poor usability is the lack of usability testing since it has been proven that conducting usability testing is critical in order to improve the usability (Nielsen, 1993; Faulkner, 2003; Raza et al., 2010). There is also proof that prolonged design discussions can stall user interface improvements by demotivating developers (Zilouchian Moghaddam, Bailey and Fu, 2012).

We start this report by describing the characteristics of OSSD. We present the definition for term *open source software*, explain the decentralized and engineer driven approach, and analyze the decision making process in OSSD. We then proceed to inspect what usability means with the help of usability heuristics. Following that we present the reasons which we think can cause usability problems in OSSD. We found out that the factors causing poor usability are numerous and can vary from insufficient usability testing to the lack of consistency in user interface designs. We conclude our report by offering a set of guidelines for avoiding some of the pitfalls of poor usability in OSSD projects. These guidelines are our main contribution in this research and they consist of five rules, which we think can be helpful to any OSSD projects if they truly intend to improve the overall usability of their products.

2. Characteristics of OSSD

In order to understand the reasons for usability problems in OSS, we should first study the characteristics of OSSD. First we describe what the term *open source software* means by briefly explaining open source licenses and how they affect the usage, modification and distribution of the source code. Then we proceed to describe the decentralized and engineering driven approach, which is typical to OSSD projects. We also depict how decisions are made in the OSSD and how project growth can affect on the decisions.

2.1 Definition of open source software

The term open source software refers to a software released under certain types of licences, providing the user the rights to use, modify and to distribute the source code of the program freely (Andreasen et al., 2006). In more detail, open source licence should require free redistribution, meaning that it shall not restrict redistribution of the software as a part of a combined software distribution, nor require any royalty or other fee. The source code in its original form must be included in the program and allow distribution in both, source code and compiled form. If not, there must be instructions of how to obtain the source code without a charge. Modifications and derivations should be allowed, as well as redistribution of these under the same terms as the original software. However, the licence may require the derived works to be named and/or version numbered differently from the original version. It may also require that the source code should be distributed as base sources and patch files to recognize the unofficial changes from the original source code. Open source licence shall not discriminate any persons, groups or fields of endeavor, everyone should be equally eligible to contribute. Software licence must apply to everyone to whom the program is redistributed and it must not be specific to a product. It should also not restrict any other softwares of possibly being redistributed along with the licensed software. In addition, open source licence should be technology neutral (Wikibooks, 2013).

2.2 Decentralized and engineer driven approach

Perhaps the most iconic example of an open source software development is the Linux operating system. It started as Linus Torvald's personal hobby project. Linus became interested in operating systems and wanted to make a free alternative to the MINIX operating system (Torvalds, 1991). Raymond (1998a) describes this kind of behavior as "scratching a developer's personal itch"; according to him, every good piece of software begins this way. In proprietary software development workers keep on coding the software because they are getting paid, not necessarily because they are interested in the software. In open source development things are somewhat different; the developers can be highly motivated simply because they will be able to make the software suitable for their own use. For this reason the average quality of open source programs tend to be good.

Although traditional face to face discussions have been proven to be more effective than synchronous or asynchronous discussions (Straus & McGrath, 1994), open source software projects very rarely utilise face to face communication because the developers can be located virtually anywhere on the planet (Raymond, 1998b). Some open source projects may adopt an approach in which a central organization manages and coordinates the work of a larger community of developers (Fitzgerald, 2006), but the majority of projects do not have formal organizational structures. Raymond (1998a) describes projects that do not have a formal hierarchy as ‘bazaars’ and continues to compare them against ‘cathedrals’ which are projects with strict hierarchies. In bazaars people keep on adding features which they themselves think are necessary or worthwhile. Cathedrals are planned carefully and the designing is done at the very top of the hierarchy.

2.3 Decision making

Having a credible leader or leadership will mean that the open source project has better chances of being successful. Often the leader is the developer who started the whole project trying to make a software for his/her own use. While this person may do most of the programming by him/herself at the beginning of the project, (s)he will most probably have less and less programming to do as the project advances. The reason for this is that the leader has a major role in decision making. Leader usually provides a vision for the project, divides the project objectives to smaller tasks, tries constantly to draw new developers to the project and in overall tries to manage the whole project so that it would not be abandoned. (Lerner & Tirole, 2002)

Raymond (1998b) coined the term “benevolent dictator” for a open source project leader who maintains the source code and has the final say in what gets included in it. If an open source project is led by this kind of dictatorship, decision making can theoretically be quite straightforward: the benevolent dictator accepts changes to the source code that in his mind benefit the software and rejects the ones that are not necessary or not up to the standards of the project. However, as the time moves on, the benevolent dictator projects have a tendency to alter their decision making process so that there will be multiple people included. Co-developers are people to whom the project leader appoints certain responsibilities such as specific subsystems. These appointed co-developers are given the right to control what will be implemented to their subsystems and in return they are committed to the maintenance of the work they control; only the benevolent dictator has the rights to make corrections to their work. Over time the benevolent dictator starts to act more like an architect than the absolute authority. When a project grows and it has managed to attract dedicated and capable developers to it, the benevolent dictator model can be discarded completely. Important decisions can be made in a committee consisting of fellow co-developers; Apache projects are good examples of this. Another applicable alternative would be rotating dictatorship, in which the position of the dictator is rotating in a group of senior co-developers. When decision making in a project is carried out by committees or rotating dictators, avoiding conflicts is quite difficult. Internal boundaries in complex projects are hard locate, because the ownership of the source code can be inconsistent.

3. Meaning of usability

Some of the OSS projects are known for good quality products. For example, Apache HTTP server, GNU/Linux operating system and Mozilla Firefox Internet browser have successfully produced good quality software when using open source methods. On the other hand some of the open source projects are known for bad user interfaces and open source world can scare users away because some OSS programs might have a variety of unconventional user interface designs they are not used to. (Benson et al., 2004; Raza & Capretz, 2010.)

Usability relates to everything in the system that a human can interact with; computer features that do not have any user interface components are uncommon. User interface's usability can not be explained with a single, one-dimensional definition. Traditionally usability is described with the help of the following attributes: *learnability*, *efficiency*, *memorability*, *errors* and *satisfaction*; by examining these measurable attributes, the usability of a software can be systematically evaluated. (Nielsen, 1993.)

One way of categorizing usability problems is by inspecting usability with heuristic evaluation. Molich and Nielsen (1990) classified user interface usability problems into 9 categories. These categories gave birth to a list of heuristics that promote good usability. Nielsen (1994a; 1994b; 1995) later refined this list and came up with 10 most general heuristics. Table 1 explains these 10 usability heuristics in greater detail.

Table 1. 10 usability heuristics (Nielsen, 1995)

Heuristic	**Explanation**
Visibility of system status	Users should be informed about what is going on in the system. The feedback should be delivered within reasonable time limits.
Match between system and the real world	System should speak language which user is familiar with. Metaphors and concepts should come from the real world so that they can be easily understood. Information should be presented in logical order imitating real-world conventions.
User control and freedom	Undo and redo options should be provided so that users could use them as emergency exits when they think they have made mistakes. Actions should be reversible.
Consistency and standards	Same things should be expressed the same way and they should look the same. System should follow platform

	conventions and not confuse the user whether or not some things mean the same.
Error prevention	Errors should be prevented from happening in the first place. Error prone conditions should be disposed of and users should be prompted with confirmation questions before committing actions.
Recognition rather than recall	Users' memory load should be minimized by offering visible objects, actions and options. Users should be able to gain al necessary information through the user interface.
Flexibility and efficiency of use	System should be efficient to use and it should provide som customization. System should provide accelerators and shortcuts so that experienced users can speed up frequent actions.
Aesthetic and minimalist design	All information that dialogues contain should be relevant. Irrelevant information can diminish the visibility of the important information.
Help users recognize, diagnose, and recover from errors	Error messages should contain precise information of the problem and suggest a solution to it. Messages should be explained in plain language without being too technical.
Help and documentation	If users need help, they should be provided with easily searchable instructions and documentation. This information should be focused on users' tasks and present step-to-step instructions without being too overwhelming.

4. Reasons behind the problems

In this chapter we search for the reasons why usability problems exist in OSS products. We delve deep into the roots of the problems by investigating the priority of usability, the importance of testing, the intended target audiences, the effect of project size, as well as the consequences of failing to reach consensus and the lack of consistency.

4.1 Low priority on usability

Raza and Capretz (2012) have conducted a study about how the developers collect users' feedback and how they meet the expectations of the end users. They surveyed the developers of different sized open source projects. According to their results, only 30% of the developers consider usability as the most important quality attribute in their projects and only 42% collected some form of user feedback for their projects. In addition, 77% of the respondents did not consult usability experts for their projects and only one-third of the respondents who consulted an expert actually followed the advices given and modified their project. (Raza & Capretz, 2012.)

Even though their sample size was quite small (72 respondents), their results indicate that the developers do not seem to perceive the importance of usability and that methods to improve the usability, such as collecting users' feedback and consulting usability professionals are not utilized in most OSS projects. Andreasen et al. (2006) also found out that usability evaluation is not considered as a priority in many OSS projects. Their study indicates that OSS developers are actually interested in usability, but in general, they do not have professional usability practices. Usability experts' evaluations are appreciated as long as they do not interfere the decision-making about changes and priorities. One reason for OSS developers not being interested in focusing usability issues is that in general, they are seeking for challenges. They want to improve their skills and want to be intellectually stimulated. (Andreasen et al., 2006.)

One reason for open source contributors not being interested in usability might be the importance of incentives. Successful tasks, such as adding a new functionality or optimising the code reward the user with the respect of the fellow contributors. Improvements made to the usability may not be considered as challenging or interesting as solving a difficult problem and thus, the contributors rather choose another area to work with. One reason for usability experts not involving to open source projects is that they do not feel welcomed to the projects. Reason behind this may be the hacker culture from which open source originates. In this kind of culture, other hackers are typically welcomed, but non-hackers, in this case the usability experts from different culture, are not appreciated (Nichols & Twindale, 2005.)

One of the major problems of any software is that they do not meet the quality requirements well enough. It can also be difficult to make the necessary changes to increase the quality of the software. Usability is important during the software development, but still many well-known software products suffer from usability issues which are not easy to fix without changing the whole software architecture. It is very expensive to fix problems on already implemented software features. Large part of maintenance costs of software systems is because of usability issues. A lot of resources can be wasted because usability requirements are not discovered until the software is already been implemented or distributed. (Folmer Gurp & Bosch 2005.)

4.2 Importance of testing

Software testing plays important role in the software development life cycle. The main objective for designing test cases is to find out errors in the software and it can make the quality of the software better. The good quality products are the ones what is wanted and appreciated. Software testing also helps to understand the risks when developing the software. The goal of testing is to check if the software does what it is supposed to do. (Ahamed, 2009).

Usability testing with real users is the most important usability method that provides direct information about how certain people use the software and reveals the problems they face when using the system (Nielsen, 1993). Raza et al. (2010) suggest having usability testing as one of the key issues on improving the usability in open source projects. Findings of their empirical study prove that usability testing and OSS usability correlate favorably. The results of their survey also suggest that formal usability testing should be taken as a part of software testing procedure.

There is no specific rule for the amount of needed usability testers to find enough usability problems from the software. When conducting usability testing, there is a 70% chance of finding about 80% of the usability problems with only five test participants. However, it is suggested to run the maximum number of test participants the schedule and budget allows, because more test users find more problems. (Nielsen, 1993; Faulkner, 2003.)

Conducting usability testing on open source software projects can be tricky, because developers can be scattered around the world and getting them in the same location may not be possible. Conducting usability testing also requires skills that some of the programmers might not have. (Saikkonen, 2004.)

4.3 Target audience

As the use of open source software increases, the amount of novice and non-technical users grows and usability becomes more and more important factor when designing a software (Raza et al., 2010). Open source developers often start working on open source projects because they want to create something for their own use (Nichols & Twindale, 2005). In the recent years OSS has gained popularity also among novice users and organizations and thus, the expectations and the requirements of the end users are no longer the same as they used to be when the end users were mainly the developers themselves (Raza et al., 2012). From developers' point of view, the user interface might seem to be just fine, while the general public might have problems using it. User satisfaction has a large role in success of both, open source and closed source proprietary software. To make the software more appealing to the general public, the developers should start developing the software from user point of view, meaning that the users' expectations and requirements are understood. (Pemberton, 2004; Raza et al., 2010.)

It should not be forgotten that users of open source software also include elderly, children and people with disabilities. It is suggested that interactive help features can increase the acceptance level of open source software amongst different user categories. (Raza et al., 2012.) In addition, culture has a significant role on software development and usability testing. Due to the fact that cultures are different, there are no formal methods to guide evaluating a product by a certain way. For example, the results from usability tests conducted in China, Denmark and India can be different. The age of usability testing moderators is not problematic for Indians but it is important to speak their language, be polite and to be respectful towards the test users. Even when moderators of usability tests are not able to see the test user, they want to know the age of the users. When the test users are female and from traditional family in India, a male member from her family has to be present during the testing and the interviews, or the moderator has to be female as well. In tests in China, the age and gender can have an effect on the results of the tests. In tests in Denmark, the test participants and moderators should be similar age, gender and have same kind of job experience. There are many things to consider when conducting usability tests with people from different cultures; age, gender, environment, infrastructure, division of labor and tools. (Clemmensen et al., 2007.) Taking into account all the usability features of an OSS product can be challenging because there can be different types of users with different technical backgrounds. In addition, people from different cultures have different needs, expectations and demands. (Raza & Capretz, 2010.)

4.4 Project size effect on usability

Since open source projects often work on small budgets due to the voluntary nature of the projects, it may not be possible to employ outside usability experts. In most open source projects, the budget does not allow having usability laboratories or setting up detailed large scale usability experiments. It has been found out that having companies involved with open source projects has increased the usability activity in these projects. However, the investment is probably not the same level as with large proprietary software developers. (Nichols & Twindale, 2005.)

To achieve good quality software, people working on OSSD have to understand communities, code modularity, project and test process management. High quality OSS products require large and sustainable community to develop quickly, debug effectively and to build new features. There have been several studies stating that building sustainable community should be the key objective in OSS projects. It has also been said that one of the success factors is having large amount of voluntary contributors in OSS projects. (Aberdour, 2007.)

Because OSS can be accessed from all over the world, there are more and more users using them and at the same time there is a better chance to get more people participating on the development of the projects. When more people are coming to the OSS projects it means that quality, measurement and management becomes harder to maintain. (Raza & Capretz, 2010.)

4.5 Difficulties reaching consensus

Long remaining usability issues are likely to exist because of developer communities' inabilities to reach consensus on user interface design decisions. Zilouchian Moghaddam, Bailey and Fu (2012) conducted a research in which they examined how mature open source software communities build consensus in user interface design discussions. They focused on distributed discussions because that is how the majority of OSS communities make their decisions; when designers and developers want to debate on usability, they typically discuss with each other via mailing lists or web forums. The data was collected from Drupal and Ubuntu developer and designer communities. The authors found out that discussions were more likely to reach consensus if the participants were experienced and had prior interaction history with each other. The authors expressed that the challenges in consensus building included: diverse perspectives over the use of the software, ownership issues and strong emotions. If reaching consensus on usability issues tends to fail, the product will not get any better and the developer community weakens making the future usability improvements less likely.

Due to the nature of usability, the decisions are hard to justify. Unlike technical issues, such as performance, usability is much more subjective and backing up solutions with solid evidence is difficult; usability testing is substantially more time consuming than e.g. measuring performance. Usability is also an area which almost everyone has opinions about. Potentially that means that more people want to get involved in the debate than in some other design areas which require more specific expertise. According to people in Ubuntu and Drupal communities, increased amount of participates in design debates usually meant difficulties in reaching consensus. (Zilouchian Moghaddam et al., 2012.)

4.6 Lack of consistency

Lack of consistency is a big problem in open source projects because developers are making decisions on usability during the development of the software and the decisions made by different developers do not always match. When developers are making decisions on the fly concerning in which environment the software runs, it can limit the amount of the possible users. This could be fixed by separating the user interface and the program logic and offering the software for multiple environments preferred by the user, rather than limiting it to a single environment. This is a lot easier to say than actually implement because it is not easy to work on different environments when they may have different logic and user interface. (Saikkonen, 2004.)

If there is more than one developer working with the user interface, especially if they work individually, it will cause the interface to be inconsistent, lowering the usability of the software. Modifying the interface in one part of the application may affect the overall consistency of the whole interface and thus, the desired changes should be carefully planned (Nichols & Twidale, 2005.)

5. Conclusions

This article shows that there are many different things to consider when thinking about usability in open source development. Thinking about usability is difficult, because people are different and they have different opinions about usability. Some of the usability problems are easier to fix than others and in some of the cases the lack of proper resources is an issue. One example of this is that in order to conduct usability tests, the community might have to hire external help. That being said, it is proven that even small amount of usability testing is better than nothing. The times are changing and OSS products are becoming more common, thus the developers are not the only end users anymore as it used to be. One problem in the open source development is that the developers are not inviting usability experts to participate in the projects. When people around the globe are using the software, it means that cultural and personal differences have to be given attention in the development of the OSS. One of the usability problems in open source is the lack of consistency, which is understandable because nowadays there are large quantities of different OSS products available from developers with varying experiences and visions in user interface design.

5.1 Five rules for better usability

In this article, we have discovered several things affecting on the usability of the software. Based on our findings, we propose these five rules for achieving better usability in OSS projects.

1. Perform usability testing. According to Nielsen (1993) & Faulkner (2003), majority of the usability problems can be revealed by having only five usability testing participants. This suggests that the usability in OSS projects could be significantly improved without having to invest in conducting large scale usability testing. That being said, it is really important to have usability testing even in small scale to improve the usability of the products. Conducting simple usability testing for few people is not very expensive and it does not require too much skills, so it should be always included on the OSS development.

2. Consider the target audience. Considering the end users is important when thinking about the usability of the product. Some people prefer to do things differently than others. One way to understand the expectations and requirements of the users' is to increase the communication between the developers and the end users. It is suggested that the requirements of the users should be considered as one key issue for improving usability in open source projects. Another key factor to improve the usability could be using incremental design approach. Novice software users tend to feel more comfortable when advanced features are introduced to them incrementally. (Raza et al., 2010.) Nichols & Twindale (2005) also suggest that the usability of open source projects could be improved by engaging typical end users into the development process. As stated by Raza et al. (2012), the users of open source software also include specific user groups, such as elderly, children and people with disabilities. Thus, it is very difficult to please everybody. We suggest that one solution might be offering customization on the software, so that the users could change the settings by their liking. It is of course very hard to actually do, but even as simple function as being able to change fonts or colors could offer satisfaction to some users and that might make the product more popular. We suggest that considering the needs of the end users can increase the usability of the product considerably to better direction.

3. Be consistent. As Saikkonen (2004) mentioned, lack of consistency is a big problem in open source software development. Our opinion is that consistency is very important and it has many aspects. For example, when the user is clicking the links on the software, it is critical to have the actions on same order on each view, or when cancel and accept buttons are switching places between the views, it can be very confusing when the action is not what it was used to be. Being consistent means also that the colors, fonts etc. will not change too much and the program does not have any strange actions which could confuse the end users. When designing completely new product it is important to inspect if there is other similar programs, how they behave and what they look like.

4. Invite experts. OSS projects should pay attention on what kind of expression they give to the people willing to contribute. According to Nichols & Twidale (2005), usability experts do not feel welcomed to OSS projects. We suggest that OSS projects should encourage usability experts to participate, for example by stating in the project website that usability experts are needed. People working on OSS development should also consult usability experts in order to make the experts to feel more desired and important in the projects.

5. Strive to form long-term user interface design teams. User interface design discussions tend to be more fruitful if the participants are experienced and have prior interaction history with each other. In Mature OSS projects user interface changes are usually decided by community voting and reaching consensus on those matters can be a complicated process. Because usability is multi-dimensional concept, it is very difficult to provide solid proof when suggesting improvements to user interface. Usability and user interface changes affect a large number of developers, each of which have their unique view on the issues at hand. Prolonged discussions wear out developers and can cause discord between them, which eventually results in development slow down. (Zilouchian Moghaddam et al., 2012.) Therefore, we suggest that OSS communities should try to form long-term user interface design teams consisting of talented people. As time goes on the teams are likely to get more efficient in their discussions, because participants get more comfortable with each other.

5.2 Limitations and the future research

Because there exists a huge amount of different kind of studies on usability problems from different authors, it is difficult to search for the meaningful ones. Our literature review only scratched the surface of the vast supply of available data on usability and OSSD. The kind of research we did would have benefitted from more systematic approach i.e. systematic literature review.

When conducting our study, we noticed that it is really difficult or even impossible to find material concerning how the amount of contributors effects on the usability in OSS development. It seems that this issue has not been studied yet in more detail, at least in large scale. It would be useful to know what kind of effect the project size has in user interface designs. Things like whether or not an OSSD community will be willing to take more risks with their designs if their project was small, and would it be better or worse to have a large community of designers and developers when innovating improvements.

We think that more future studies should be focused on how the usability problems could be avoided in OSS products. It seems that a major contributor to poor usability is the lack of interest to invest resources to user interface improvements. Should OSS developers be given some kinds of rewards when they invest in the usability of the software, if yes which kind of rewards would they prefer? It would be interesting to know exactly what kind of incentives would be most efficient. Also knowing about how OSS projects have improved usability during the time would be interesting to see and at the same time how the amount of money actually affects on the usability. One very important research would be, is it beneficial for the company if they are spending more time to improve the usability of the software.

References

Aberdour, M. (2007). Achieving Quality in Open-Source Software. IEEE Software, 24(1), 58–64.

Ahamed, S.S.R. (2009). Studying the feasibility and importance of software testing: An analysis. *International Journal of Engineering Science and Technology*, 1(3), 119-128.

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*, 35A(3), 303-312.

Benson, C., Muller-Prove, M. & Mzourek J. (2004). Professional usability in open source projects: Gnome, openoffice.org, netbeans. *Extended abstracts of the 2004 conference on Human factors and computing systems*, 1083–1084.

Clemmensen, T., Shi, Q., Kumar, J., Li, H., Sun, X., & Yammiyavar, P. (2007). Cultural Usability Tests - How Usability Tests Are Not the Same All over the World. *Proceedings of the 2nd International Conference on Usability and Internationalization*, 281–290.

Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. Behavior Research Methods, Instruments, & Computers. Volume 35, Issue 3, pp 379-383.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Q.*, *30*(3), 587–598.

Folmer, E., Gurp, J.V. & Bosch, J. (2005). Software Architecture Analysis of Usability. *Proceedings of the 2004 International Conference on Engineering Human Computer Interaction and Interactive Systems,* 38–58.

Lerner, J., & Tirole, J. (2002). Some Simple Economics of Open Source. *The Journal of Industrial Economics*, *50*(2), 197–234.

Nichols, D. M., & Twidale, M. B. (2005). The usability of open source software (originally published in Volume 8, Number 1, January 2003). First Monday, 0(0). Retrieved 2.11.2013, from http://firstmonday.org/ojs/index.php/fm/article/view/1494

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 249–256.

Nielsen, J. (1993). *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Nielsen, J. (1994a). Enhancing the explanatory power of usability heuristics. Proc. ACM CHI'94 Conf.(Boston, MA, April 24-28), 152-158.

Nielsen, J. (1994b). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY.

Nielsen, J., (1995). 10 Usability Heuristics for User Interface Design. Retrieved 2.11.2013, from http://www.nngroup.com/articles/ten-usability-heuristics/

Pemberton, S. (2004). Scratching Someone else's Itch: (Why Open Source Can't Do Usability). interactions, 11(1), 72–72.

Raymond, E. S. (1998a). The cathedral and the bazaar. *First Monday*, *3*(2). Retrieved 2.11.2013, from http://journals.uic.edu/ojs/index.php/fm/article/view/578

Raymond, E. S. (1998b). Homesteading the Noosphere. *First Monday*, *3*(10). Retrieved 2.11.2013, from http://journals.uic.edu/ojs/index.php/fm/article/view/621

Raza, A. & Capretz, L. F. (2010). Contributors' preference in open source software usability: An empirical study. *International Journal of Software Engineering & Applications*, 1(2), 45-64*.*

Raza, A., Capretz, L. F., & Ahmed, F. (2010). Improvement of Open Source Software Usability: An Empirical Evaluation from Developers Perspective. *Advances in Software Engineering*, 2010(2), 1-12.

Raza, A. & Capretz, L. F. (2012). Do open source software developers listen to their users?. *First Monday*, 17(3). Retrieved 2.11.2013, from http://firstmonday.org/ojs/index.php/fm/article/view/3640/3171

Raza, A., Capretz, L. F., Ahmed, F. (2012). Users' perception of open source usability: an empirical study. *Engineering with Computers*, 28(2), 109-121.

Saikkonen, K. (2004). *Usability in Open Source Projects* [pdf]. Retrieved 2.11.2013, from SoberIT, http://www.soberit.hut.fi/T-76.5651/2004/english/papers04/Final-05-Saikkonen.pdf

Straus, S. G., & McGrath, J. E. (1994). Does the medium matter? The interaction of task type and technology on group performance and member reactions. *The Journal of applied psychology*, *79*(1), 87–97.

Torvalds, L.B. (1991, August 26). What would you like to see most in minix? [Msg 1]. Message posted to https://groups.google.com/forum/#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ

Wikibooks. (2013). Open Source. Retrieved 23.10.2013, from http://en.wikibooks.org/wiki/Open_Source

Zilouchian Moghaddam, R., Bailey, B., & Fu, W.-T. (2012). Consensus building in open source user interface design discussions. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1491–1500). New York, NY, USA: ACM. doi:10.1145/2207676.2208611