International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 02– Issue 05, September 2013

The Usability of Free/Libre/Open Source Projects

A Review

Ljiljana Despalatović
The University Department of Professional Studies
University of Split
Split, Croatia
Email: ljiljana.despalatovic {at} gmail.com

Abstract—**The open source software is widely spread, yet still unpopular in the personal computer market. The reason for that could be the common opinion that the open source systems are created for advanced users, and that ordinary users do not have the time to learn how to use different user interfaces. Based on our selection and analysis of scientific articles we tried to find the answers to the following questions: is there a usability problem in open source projects, and what could be the solutions? The root of the problem seems to lie in the open source development process itself, where the focus is not placed on the user or usability, but on the system or the programmer.**

Keywords-HCI;FLOSS;usability

I. INTRODUCTION

Ever since it was crated, the UNIX operating system has been considered "complicated to use." Donald Norman's famous text, "The trouble with UNIX: The user interface is horrid" [1] sealed the fate of the UNIX operating system as unusable for an average user. It is interesting that, even though it was written in 1981, it is still taken as a reference when talking about the usability of UNIX and similar systems.

The UNIX has come a long way since then. It had started as free software, and in time became proprietary and less available to the software community. To fill the gap, Richard Stallman started the GNU project with the idea to provide an operating system with a similar functionality and interface as the UNIX has, though completely rewritten. Out of the need to protect the right to modify and redistribute software, *copyleft* licenses were created, and software released under such licenses began to be called Open Source Software (OSS) or Free Software. Since recently, a frequently used abbreviation is FLOSS (Free/Libre/Open Source Software).

The most prominent examples of FLOSS software are GNU/Linux operating system, Apache HTTP server, Mozilla Firefox web browser, MySQL, Libre Office and Android operating system.

The FLOSS has gained a reputation as a reliable, efficient, transparent and functional software. The expansion of the Internet has enabled developers worldwide to participate in superior FLOSS projects within their area (e.g., Apache web server) [2]. However, most users of FLOSS have a technological background, while the average user is still using proprietary solutions. One of the reasons for that is the common notion that the FLOSS systems have poor user interfaces [3].

The objective of this article is to select and review published literature and to answer the following research question: *What are the main research themes in scientific literature regarding the usability problem in FLOSS?*

The remainder of this article is organized as follows. The second chapter lays out the basic principles of the open source software development and explains the life cycles of FLOSS projects. In the third chapter we provide our selection of articles from the relevant databases, investigating the usability of FLOSS projects and possible approaches to improve the usability aspects of a project. Major challenges recognized in the literature deal with the user-centred model in FLOSS, using developers' motivation and human computer interaction (HCI) experts for the improvement of usability, and development of tools for evaluation of usability. Other challenges and approaches are described in the last section of the chapter. Finally, the last chapter concludes the article.

II. FLOSS CHARACTERISTICS

A. The FLOSS development model

The FLOSS development model fundamentally differs from the classical development model of software engineering. From the perspective of usability it has a two-fold meaning. On the one hand, insufficient attention is paid to the user interface design and usability, and, on the other, the project development is such that the user can easily be involved in all stages of development. That gives a great opportunity for the FLOSS projects development, and also for improving usability and design.

A FLOSS project usually starts as an early release of a product by an individual programmer who tries to gather a community of supporters that would later on join the development team. In most cases, there is no plan or specification of the project. The community, if gathered, communicates via the Internet and members do not know each other. Majority of FLOSS programmers (88%) are hobbyists [4] who have other primary jobs.

Projects rarely start from scratch. In the open code community the usual way to start a project is to legally download someone else's project and continue its development

in a different direction than the original. The alternative project is called *fork*.

After the first release of the software, the software testing community that communicates through mailing lists and communication tools provides comments, bug reports and fixes. Their feedback is then being considered and, if accepted, implemented in the next *beta release*. The process continues until the project members announce a stable release. After that, the project development process continues to the next stable version.

It is a surprise that the FLOSS projects are so successful since they violate the rules of traditional software engineering, where well-managed projects are planned in advance by collecting user requirements and creating design specifications prior to the beginning of implementation. In contrast, FLOSS projects include early coding relying on the community, which will continue with the development and design of the project.

B. Lifecycle

The FLOSS project lifecycle is divided into several phases: Introductory Phase, Growth, Maturity, Decline or Revive [5]. In the *Introductory Phase* a programmer or a group of programmers create the initial version of a software. The project is then registered at one of the web-based hosting services for software development projects, such as Sourceforge.net, Github, or similar, which allows other developers to engage in software development. At this stage, the task of the programmer-project manager, *maintainer*, is to create the initial version of the software, assemble a team and roll out the vision of the project's future. However, the project manager does not allocate tasks to other developers, but they choose jobs according to their interests.

The project *grows* when users recognize and accept it. Administrative obligations are growing along with the growth of the project: analysis of reported errors, customer support, and change requests. New roles are assigned to the programmers, beta versions are released, but the initial team is still controlling the project, while other developers take less important functions.

At the *maturity* stage, the project reaches a critical mass. Project managers delegate programmers, evaluate source code and determine the direction of the project development. The majority of decisions are made by consensus, but in case of conflicting opinions within the community, the project managers take the role of moderators. A motivation for the project becomes weaker, and part of the community takes upon themselves to motivate new members.

Finally, the project can reach the stage where the project manager has fulfilled his or her plans and leaves the project. If in such a case someone else is interested in this role, the project *revive*s. Otherwise, the project *declines*. Often, a part of the community does not agree with the direction in which the project manager is leading the project. In such a case, the project branches and the community moves on to a forked project.

C. Roles

The FLOSS projects do not only involve programmers. There are many jobs that can be performed by people with different interests and technical skills: testing and reporting errors, requirements for new functionality, translation, helping other customers, making illustrations and artwork, writing documentation, and collecting donations. The exact mechanisms for participation in the FLOSS project depend on the governance model. There are two basic governance models: *benevolent dictatorship model* and *meritocratic model* [6].

Benevolent dictators are generally people who started the project. Their role is to determine the general guidelines of the project, and to resolve any disagreements within the community. In small communities the benevolent dictator can decide in a similar way as in the classical management model, while in larger projects he or she usually takes the role of an arbitrator and coordinator. Examples of such management roles are Linus Torvalds in the Linux kernel project, Guido van Rossum in the Python project, Rasmus Lerdorf in PHP project, and Larry Wall in Perl project [7]. As they became famous in the FLOSS world, their role was renamed to "Benevolent Dictator for Life".

In the meritocratic management model, management roles are assigned by excellence, and not by social or political status. This does not mean technological superiority, but the ability to balance between the different constraints: trends, customer needs and available resources [8].

Other notable roles in FLOSS projects are *contributors*. They participate in FLOSS projects in many different ways: activities on forums and mailing lists, and active participation on public bug system, where they can report errors and feature requests, create patches, report duplicate requests or errors, etc. There is also a subset of contributors called the *committers*: the developers who can modify the code.

III. THE USABILITY OF FLOSS PROJECTS

Despite being rather unusual, the FLOSS development model has introduced new concepts in software engineering, and proved to be sufficiently tough to have grown stronger over the years. However, although the FLOSS principles themselves do have a perspective, a wide range of users still does not fully accept FLOSS projects. What is the reason for that? Why the free operating system Linux has never succeeded to knock down the widespread and in many ways inferior MS Windows operating system. Why are Mozilla Firefox and Chrome Web Browser still struggling with Internet Explorer? Why do people buy the expensive Photoshop, when there is the easily accessible GIMP? Why do users write documents in MS Word? In this chapter, we will try to provide answers to these questions.

It should be noted that usability is not the only reason for not using the FLOSS solutions. Some of the credit should be given to poor marketing or no marketing at all. Then there are hardware manufacturers having contracts with the proprietary software companies. The lack of understanding of licence policies is also a barrier [9]. But the greatest of all is a widespread opinion that the user interface is not good enough for nonprogrammers. The contribution to this opinion gave Donald Norman in his article [1], where he had rated UNIX as unusable. UNIX is no longer considered to be free software, but Linux inherited UNIX principles and also its usability assessment. As a renowned UNIX hater, Norman was invited to write the foreword to the book "UNIX-HATERS

handbook" [10]. The preface ended with the words "As for me? I switched to the Mac. No more grep, no more piping, no more SED scripts. Just a simple, elegant life: Your application has unexpectedly quit due to error number -1. OK?"

However, years have passed since and the situation has changed. At the time of its creation, free software was designed for programmers. Today, it is not so. The usage of the FLOSS projects increases rapidly with the growth and availability of the Internet. The FLOSS is not a "reserved arena" for technologically educated users; beginners and technologically untrained users around the world use FLOSS solutions.

The research [11] conducted among 106 developers from 18 different FLOSS projects showed that 87% of developers believe that the user's requirements improve the usability of FLOSS. On the other hand, there is no positive association towards the inclusion of HCI experts (usability experts' opinion). And as the most important attribute towards usability improvement is considered to be an incremental design approach and usability testing. All of the participants in the survey agreed with the statement that computer science students should be aware of the aspects of user-centred software design and to have full knowledge of the UCD methods.

The usability is described by five characteristics: learnability, efficiency, memorability, low user error rate, and users' subjective satisfaction [12], and should be separated from the questions of reliability, functionality and efficiency of the program itself. Usability of the programs intended for developers, such as compilers or source code editors, are not included in the usability survey. Ordinary users have no technological background, do not have the patience and they want to have the whole of functionality in just "two mouse clicks". The usability is not a technical problem, but a problem of ergonomic design and user interface psychology, and developers have always been bad at it [13]. An unprofessional user can hardly be attracted by the availability of the source code and his/her choice of software solutions will more likely be based on other criteria.

A. The FLOSS and User-centred Design

The basic idea behind the user-centred design is user involvement in all phases of software development: analysis, design, implementation and deployment. Basic principles of FLOSS projects are in line with that: to allow the users to use, change and distribute software means to allow the user the ability to participate in all stages of development. However, in the FLOSS development model that consists of early prototypes, frequent software updates and an active community of testers, not only that the user participates in the design and testing of the software, but also in assessing the usability once the project is released, in the post-deployment stage. Users participating in that way are *active* users. Beside them, there is a large community of *passive* users, who do not participate in the software development process [14].

The HCI as a discipline tends to underestimate the importance of the software usability testing at the post-deployment stage [15]. From the standard model of software engineering it is known that it is easier and cheaper to correct errors in the early stages of software development. Most HCI experts emphasized the need to incorporate the HCI method in the specifications phase and prototypes production phase. In standard development models of the proprietary software that makes sense: once the product is shipped, the communication with the user is weak. Such software release cycles are long and the user has a chance to evaluate the product until the next release, but the waiting time for changes is long [16].

Nichols [14] highlights several trends in the software development that allow the adding of usability test methods in the development phase after the release of the product. These are the ease of establishing communication between the users and the development team via the Internet, incremental versions of the software, and the ease of upgrading existing software.

In a study of user frustration [17] it was observed that one third to one half of the time the users spend in front of a computer is lost due to frustrating experiences. The question posed by Nichols is: what a computer program should do in "moments of frustration"?

He provides several possible answers. The first answer is the standard answer: do nothing. Most of the today's software is passive, instructing the user to use the system for help and to wait for the next version. The affective response (the second answer) describes the software's ability to respond to emotional states of the user. The third answer is the engineering response, which means that programs should be made more flexible, configurable, i.e. to change the way it interacts with the user.

The last two responses give the user a more active role in software development, by proposing to the user to improve the source code or enabling communication between users and the development team. The former is a FLOSS solution and the latter the "*post-deployment*" usability. The limiting factor of the FLOSS solutions is possible technological ignorance of users. One article [18] stresses the need for easier communication with users, by involving them in the design process and providing them with tools for easier troubleshooting. Bach and Twidale [19] propose the use of reflective-user reports and sharing of reflections, and the elaboration of these reflections by social network platforms. They call these users reflective users.

B. Motivation for Participation in FLOSS Projects

Although it seems that the FLOSS development model is democratic and user-centred, it is actually a system-centred model. FLOSS users generally are developers themselves or technically educated users. Such users value functionality, speed and the beauty of the source code above usability.

The researches in programmers' motivation [4, 20, 21] investigate the motivation of developers to participate in FLOSS projects.

There is an internal (cognitive) and an external (social) motivation for participation in FLOSS projects [22]. The internal motivation is learning, while the external are social status and reputation in the FLOSS community. Moreover, the main motivation of programmers is ideology. In one survey [21], the most commonly selected answer (88%) to the question of motivation was "To strengthen the free software". However, to answer the question about the motivation to improve usability, one of the participants in the survey replied:

"... hackers are programming for fun, and there is certainly more fun in adding support for a protocol, than repairing dialogues for Grandma."

The attitude towards the customers is also the attitude towards the usability of software. Programmers often have prejudices against inexperienced users and call them "*simple users*" or "*stupid users*", as opposed to the terms used for technical users, "*advanced user*" or "*power user*" [4]. While pointing out the ideals of freedom and cooperation, FLOSS developers do not appreciate the views of stakeholders outside the technologically advanced users.

HCI experts are not motivated in the same way as hackers and typically they do not feel welcome to the FLOSS community [23]. One way to change that is to educate developers about the basic usability principles. To avoid disagreements with the (re)design done by HCI experts, the usability analysis and design rationale could convince developers of the need for user-centred design, as opposed to their system-oriented design. On the other hand, FLOSS could be an interesting area of research for HCI students. But FLOSS projects are mostly done by volunteers or have a small budget so they can hardly attract HCI experts.

In order to find a suitable way to introduce usability activities in a FLOSS project, one study [24] compared two approaches, a consultative role and a participative role of usability specialists. The study suggested that a usability specialist should adopt the participative role and become a member of the team, meaning that he or she should understand the philosophy and the development process of FLOSS and play by the rules of community, while promoting the interests of nontechnical users.

C. Tools for usability evaluation

FLOSS projects use in their development a variety of tools: compilers, mailing lists, version control systems (CVS), memory leak detectors, repositories, etc. However, except for the tools for rapid development of user interfaces, they do not use any tools for evaluating usability. The HCI community does not have well-developed tools for automatic evaluation of non-web applications [25]. Besides that, the FLOSS community developers are willing to use only FLOSS tools. A research [26] conducted on a dataset consisting of 1753 FLOSS projects showed that online forums play a significant role in identifying and fixing usability bugs. Although online forums could be important for gaining feedback from users, they are not suitable for error reporting. Usability bugs handled by mailing lists and forums are hard to follow [27], and should be moved to a bug tracker system to be recognized and resolved by developers.

Bug reporting systems are not customized for reporting usability errors. Adapting such systems for HCI experts would facilitate their use. For example, to allow screenshot images in the text. Although it may appear trivial, the main lesson learned from usability studies is that details are essential and a small amount of extra effort is sufficient to distract users from participation [12]. Error reporting systems (e.g., Bugzilla) are complicated for the average user, and require registration and a specific amount of time to learn the user interface. Crash reporting tools would be much more useful.

As for the interface design and evaluation by users, the shift has been made to facilitate the users to report usability errors. Two methods stand out: the design-by-blog, like Firefox Input Dashboard (formerly Hendrix), and distributed heuristic evaluation through the bug reporting tool (Bugzilla) [28]. Such tools can be used for post-deployment evaluation of the project.

Customization of the error reporting tools in distributed heuristic evaluation could change the way developers see the problem of usability. The FLOSS community uses a dictionary to explain the good and bad things appearing in the project, such as performance, crashing, and data loss. Expanding the dictionary with words related to usability, like consistency and feedback, would facilitate the naming of the problem and connecting similar problems. Typically, usability issues in error reporting systems are delineated descriptively. Faaborg [28] described the integration of HCI principles in the Bugzilla project, the bug tracker system used by FLOSS communities. It was performed by defining 17 heuristic principles, with keywords assigned to them, which enabled programmers to easily identify usability problems and connect them with other errors (and solutions) of the same type.

D. Other approaches

Nichols and Twidale [20] suggest more approaches in order to maximize usability.

- **The commercial approach** is the inclusion of large companies in development projects. It is a common practice in the FLOSS world. Large companies already participate in FLOSS projects by paying developers and performing usability testing [29].
- **Involving the end user**. **Creating a usability discussion infrastructure**. **Fragmenting usability analysis and design**. In a similar way to how the FLOSS development divides a project into smaller pieces, usability testing can also be fragmented so that only a certain number of people around the world, each of whom doing a usability study, combine their work into a common result. It is surprising how much information about the usability can be drawn from a small number of such studies [12].
- **Education and evangelism**. As commercial software producers need to learn that the usability is essential to sell their products, the FLOSS community should also need to be educated about the importance of usability. Although sales and earnings in FLOSS projects hardly play any role, the distribution of the software does. Large user base is the common motivation of FLOSS developers. On the other hand, it is also important that the HCI experts feel welcome in the FLOSS community so as to be able to communicate with each other productively.

Recommended guidelines to be followed in the usability of FLOSS in [18] include developing the standard for usability evaluation. A reconceptualization of HCI methods to better fit the FLOSS culture is proposed in [30]. The opposite of that is the model proposed in [31]. The model extends the role structure in software development model of FLOSS by adding

the level of communications and roles. Existing technical level consists of technical core team, committers, contributors, and active users. An additional human level consisting of HCI core team, usability designers, usability evaluators and nontechnical users will be equally important in the development process. Both approaches, the HCI and FLOSS software development model reconceptualizations are a huge shift in their own fields.

IV. CONCLUSION

The FLOSS community is one of the largest collaborations in the world that has produced much of the existing software. However, it is not sufficiently recognized by software engineering, or by HCI experts, so it is not the subject of a large number of studies. But, the question of usability in FLOSS projects has been opened and suggestions are given for improving usability.

In this review, articles are divided in four groups based on their focus concerning the usability problems in FLOSS and suggestions for improvement. The first group includes studies that emphasize the necessity of the shift from the system-centred design to the user-centred design. The next group focuses on developers and HCI experts' motivation for participation in FLOSS projects and investigates the relationship between these two communities. The third group consists of studies that emphasize the importance of automatic usability testing tools. And finally, the last group gives us a few more suggestions how to improve usability in FLOSS projects.

The purpose of this work is to motivate HCI researchers to put more emphasis on the FLOSS project and to motivate developers to put more emphasis on the usability of their projects.

With the increased number of users participating in the development of FLOSS projects, the prospects for the improvement of usability have risen. If FLOSS communities enable channels for easier communication between users and developers, FLOSS could make a breakthrough with regard to the issue of usability, as it has already made in terms of functionality and reliability.

REFERENCES

[1] D. Norman, "The trouble with unix, 139-150," *Datamation, Nov*, 1981.

[2] D. Nichols and M. Twidale, "The usability of open source software," *First Monday*, vol. 8, no. 1, 2003.

[3] C. Benson, M. Muller-Prove, and J. Mzourek, "Professional usability in open source projects: Gnome, openoffice. org, netbeans," in *CHI'04 extended abstracts on Human factors in computing systems*. ACM, 2004, pp. 1083–1084.

[4] D. Yeats, "Open-source software development and user-centered design: a study of open-source practices and participants," Ph.D. dissertation, Texas Tech University, 2006.

[5] D. E. Wynn, "Organizational structure of open source projects: A life cycle approach," in *Abstract for 7th Annual Conference of the Southern Association for Information Systems, Georgia*, 2003.

[6] G. H. Ross Gardler, "Meritocratic governance model," URL: http://www.oss-watch.ac.uk/resources/meritocraticGovernanceModel.xml, Jun. 2010.

[7] J. Malcolm, *Multi-stakeholder governance and the Internet Governance Forum*. Consumers International, 2008.

[8] E. Lee, "Open source development: The diversocracy," URL: http://www.redhat.com/magazine/016feb06/features/meritocracy/, Tech. Rep., 2006.

[9] S. Walli, D. Gynn, and B. Rotz, "The growth of open source software in organizations," *A report*, 2005.

[10] S. Garfinkle, D. Weise, and S. Strassmann, "Unix-hater handbook," 1994.

[11] A. Raza, L. F. Capretz, and F. Ahmed, "Improvement of open source software usability: an empirical evaluation from developers' perspective," *Advances in Software Engineering*, vol. 2010, 2010.

[12] J. Nielsen, *Usability engineering*. Access Online via Elsevier, 1994.

[13] E. Raymond, *The cathedral and the bazaar*. Springer, 1999, vol. 12, no. 3.

[14] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005.

[15] D. M. Nichols, D. McKay, and M. B. Twidale, "Participatory usability: supporting proactive users," in *Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction*. ACM, 2003, pp. 63–68.

[16] P. K. Chilana, A. J. Ko, J. O. Wobbrock, T. Grossman, and G. Fitzmaurice, "Post-deployment usability: a survey of current practices," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 2243–2246.

[17] K. Bessiere, I. Ceaparu, J. Lazar, J. Robinson, and B. Shneiderman, "Understanding computer user frustration: Measuring and modeling the disruption from poor designs," Tech. Rep., 2003.

[18] M. C. Yelleswarapu, "An assessment of the usability quality attribute in open source software," 2010.

[19] P. M. Bach and M. Twidale, "Involving reflective users in design," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 2037–2040.

[20] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," pp. 419–429, 2003.

[21] M. Andreasen, H. Nielsen, S. Schrøder, and J. Stage, "Usability in open source software development: opinions and practice," *Information technology and control*, vol. 25, no. 3A, pp. 303–312, 2006.

[22] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[23] D. Nichols and M. Twidale, "The usability of open source software," *First Monday*, vol. 8, no. 1, 2003.

[24] M. Rajanen, N. Iivari, and K. Anttila, "Introducing usability activities into open source software development projects–searching for a suitable approach," *Journal of Information Technology Theory and Application (JITTA)*, vol. 12, no. 4, pp. 5–26, 2011.

[25] D. M. Nichols and M. B. Twidale, "Usability processes in open source projects," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 149–162, 2006.

[26] A. Raza, L. F. Capretz, and F. Ahmed, "Usability bugs in open-source software and online forums," *IET software*, vol. 6, no. 3, pp. 226–230, 2012.

[27] G. Çetin, D. Verzulli, and S. Frings, "An analysis of involvement of hci experts in distributed software development: practical issues," in *Online Communities and Social Computing*. Springer, 2007, pp. 32–40.

[28] A. Faaborg and D. Schwartz, "Using a distributed heuristic evaluation to improve the usability of open source software," 2010.

[29] P. M. Bach, R. DeLine, and J. M. Carroll, "Designers wanted: participation and the user experience in open source software development," in *Proceedings of the 27th international conference on Human factors in computing systems*. ACM, 2009, pp. 985–994.

[30] M. Terry, M. Kay, and B. Lafreniere, "Perceptions and practices of usability in the free/open source software (foss) community," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 999–1008.

[31] H. Hedberg and N. Iivari, "Integrating hci specialists into open source software development projects," in *Open Source Ecosystems: Diverse Communities Interacting*. Springer, 2009, pp. 251–263.