Pontifícia Universidade Católica
do Rio de Janeiro

Fabiana Pedreira Simões

Supporting End User Reporting of HCI Issues in Open Source Software Projects

DISSERTAÇÃO DE MESTRADO

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Profa. Simone Diniz Junqueira Barbosa

Rio de Janeiro
August 2013

Pontifícia Universidade Católica
do Rio de Janeiro

Fabiana Pedreira Simões

Supporting End User Reporting of HCI Issues in Open Source Software Projects

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Profa. Simone Diniz Junqueira Barbosa
Advisor
Departamento de Informática – PUC-Rio

Profa. Clarisse Sieckenius de Souza
Departamento de Informática – PUC-Rio

Prof. Julio Cesar Sampaio do Prado Leite
Departamento de Informática – PUC-Rio

Prof. José Eugênio Leal
Coordinator of the Centro Técnico Científico da PUC-Rio

Rio de Janeiro, August 21st, 2013

Fabiana Pedreira Simões

Graduated in Information Systems from PUC-Rio in 2011. She is an active contributor in the GNOME Project, an open source desktop environment. Her main interest areas are Human-Computer Interaction and Open Source Communities.

Bibliographic data

Simões, Fabiana Pedreira

Supporting end user reporting of HCI issues in open source software projects / Fabiana Pedreira Simões ; advisor: Simone Diniz Junqueira Barbosa. – 2013.
147f : Il. (color.) ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2013.
Inclui bibliografia

1. Informática – Teses. 2. Relatos de problemas de IHC. 3. Comunidades open source. 4. Avaliação de IHC. I. Barbosa, Simone Diniz Junqueira. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my mother.

Acknowledgments

To my mother, whose support and love brought me here.

To Simone Barbosa, the best adviser one could have, who is both an inspiration and an example to me. Thank you for the support, patience and brownies.

To Lorena Miguel, who has been coping with me for six years. Thank you for always trusting me, regardless of how much I distrusted myself.

To Andreas Nilsson, whose love and uplifting attitude comfort and lighten me. Thank you for always believing in this work. Jag älskar dig.

To Vinicius Segura, José Antônio Motta, and Taíssa Abdalla, who helped me laugh at tragedy. Thank you for the support and for making this all easier.

To the GNOME community, for embracing me and giving me an opportunity to contribute to a better world.

To CNPq, who supported and financed this work.

Abstract

Simões, Fabiana Pedreira Simões; Barbosa, Simone Diniz Junqueira (Advisor). **Supporting End User Reporting of HCI Issues in Open Source Software Projects**. Rio de Janeiro, 2013. 147p. MSc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Empowering end users to proactively contribute to OSS by reporting HCI issues not only represents a potential approach to solving HCI problems in OSS projects, but it also fits the Open Source values and ideology. By referring to the end users' personal experiences and feedback reports, designers in OSS communities may build their design rationale not only in an open and transparent manner, but also in such a way that end users relate to the values embedded in the community. This study aims to contribute to the existing literature by exploring (a) how issue reports fit and influence OSS designers' activities, (b) what the information needs of designers in OSS projects are, and (c) how to support users on the task of creating HCI issues reports that meet those needs. In order to collect data about questions (a) and (b), we conducted interviews with four designers contributing to OSS projects, and qualitatively evaluated a set of 547 bugs reported under HCI-related keywords. Using this data and based on Semiotic Engineering, we designed a form for users to report HCI issues. To investigate how well this form communicates the information needs of OSS designers and address question (c), we performed a study in which participants were invited to report HCI issues through the designed form.

Keywords

End user reporting of HCI issues; open source communities; HCI evaluation

Resumo

Simões, Fabiana Pedreira Simões; Barbosa, Simone Diniz Junqueira. **Apoiando o Relato de Problemas de IHC em Projetos de Software Open Source**. Rio de Janeiro, 2013. 147p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Capacitar usuários a contribuir proativamente para projetos de Open Source Software (OSS) através de relatos de problemas de Interação Humano-Computador (IHC) não apenas representa uma alternativa para resolver problemas de IHC em projetos de OSS, mas uma que se enquadra nos valores e ideologia da comunidade OSS. Através de referências às experiências comunicadas através de relatos de problemas de IHC, designers em projetos de OSS podem não apenas fazer suas decisões de design de uma maneira aberta transparente, mas também de uma maneira que os usuários finais possam se relacionar com os valores da comunidade. Esse estudo tem como objetivo explorar (a) como relatos de problemas se encaixam e influenciam as atividades de designers em projetos de OSS, (b) quais são as informações necessárias dos designers em projetos de OSS, e (c) como podemos apoiar os usuários na criação de relatos que estejam alinhados com essas informações. Para endereçar as questões (a) e (b), nós realizamos entrevistas com quatro designers contribuindo para projetos de OSS, e avaliamos qualitativamente um conjunto de 547 bugs reportados com palavras-chave relacionadas à IHC. Com esses dados, nós elaboramos um formulário, com base na Engenharia Semiótica, para usuários

relatarem problemas de IHC. Para avaliar quão bem esse formulário comunica as informações que designers em projetos de OSS precisam e, então, endereçar a questão (c), nós conduzimos um estudo onde participantes foram convidados a reportar problemas de IHC através do formulário elaborado.

Palavras-chave

Relatos de problemas de IHC; comunidades open source; avaliação de IHC

Contents

List of Figures

List of Tables

It's free as in freedom.

1 Introduction

The Open Source Initiative[1] (2012) defines open source as a "development method for software that harnesses the power of distributed peer review and transparency of process". Stallman (1992) endorses this view by referring to open source as a "cooperative endeavor that produces a shared resource for all." Also, according to Ghosh (2006), open source software (OSS) is software developed through a collaborative, informal network of professional or amateur developers who subscribe to the open source ideology. Because of its distributed nature, OSS development happens mostly over the internet, making OSS projects of easy access to anyone interested in joining or using them.

One of the characteristics of OSS development is that anyone, user or contributor, can report problems found while using a piece of OSS. Raymond (1998) compared the development styles of open and closed source software projects. Based on this, he described some key defining aspects of OSS development, the most famous of which being "given enough eyeballs, all bugs are shallow". Also known as Linus' law, this quote refers to how, by having a

1 http://opensource.org/

large number of community members using and testing software, problems are found, reported and fixed. This means that software problems get characterized very rapidly within OSS development.

From the perspective of Human-Computer Interaction (HCI) research, this dialog, in the form of reports from user communities to developers,[2] presents a promising opportunity for the identification of interface and interaction issues in software. In the context of OSS, this is an important goal to achieve, given the fact that OSS is widely considered to have low usability and poor user experience (Nichols and Twidale, 2003). In fact, we find increasing the involvement of users through reports of HCI issues among the potential approaches proposed by Nichols and Twidale (2006) to overcoming HCI problems in OSS. This dialog between users and designers[3] through reports of HCI issues is also known as "user-reported incident" (Castillo *et al.*, 1998; Hartson and Castillo, 1998), one of the most widely known approaches to post-deployment research, a subset of the remote research methods in HCI (Hammontree, 1994).

There are three trends in OSS development that make use of post-deployment research suitable to OSS projects (Nichols *et al.*, 2003): low-cost reliable networking connecting users and developers, incremental software versions, and easy upgrades for users. In addition, Hartson and colleagues (1996)

2 In this context, "developers" include all contributors in an OSS project, regardless of their involvement with code.

3 In this work, we use the term "designers" to refer to HCI designers only.

claim that processing the results of other remote methods, such as user journals or automatically generated logs of user actions, can be like "looking for a needle in a haystack". This can be a problem in OSS development, since a large portion of its developers contribute on their spare time. However, as Hartson also observes that "if users can deliver the needles directly, we can avoid the haystack", in reference to the user-reported incident method.

A number of works on OSS development has focused on incident reporting from the perspective of correctness of behavior and functionality. However, little research has focused specifically on the matters of user involvement with these reports. The studies most closely related to this topic observe interactions between power users and developers through bug reporting systems (Bettenburg *et al.*, 2008; Ko and Chilana, 2010). Other researches focus mostly on using bug reporting systems as a data point to study coordination aspects of open source communities (Sandusky and Gasser, 2005; Li *et al.*, 2008) and the transitions users make until they become active developers (Von Krogh *et al.*, 2003; Ducheneaut, 2005; Herraiz *et al.*, 2006; Jensen and Scacchi, 2007).

The topic of user reporting of HCI issues has also received relatively little attention in the research community. Previous work (Frishberg *et al.*, 2002; Nichols *et al.*, 2003; Nichols and Twidale, 2003; Raymond, 1999) has suggested that HCI issues are not easily dealt with in OSS projects. Indeed, Wilson and Coyne (2001) debate whether HCI issues actually belong to the same system as issues related to code. In response to that, a few works raise discussion on how

HCI-related bugs are managed in OSS projects (Sandusky *et al.*, 2004b; Scacchi, 2002) and on how HCI discussions take place within OSS bug reporting systems (Twidale and Nichols, 2005). Apart from the literature on remote methods, the work most closely related to the specific topic of user reporting of HCI issues mostly states the potential of the technique for OSS projects (Nichols *et al.*, 2003; Nichols, 2003).

Empowering and motivating users to proactively contribute through the reporting of HCI issues not only represents a potential approach to the problem of HCI activities in OSS projects. It also fits the open source values and ideology, since it poses a way by which designers may inform the rationale of their design decisions in an open and transparent manner. From our involvement in the GNOME[4] project, this is a critical point for OSS communities to trust and, therefore, value designer participation.

OSS communities have shown the potential of its distributed model to achieve rapid results in terms of functionality. As Nichols (2003) also asks, can we take advantage of this model to leverage HCI activities and create easier feedback channels for the less technical user? This study aims to contribute to the existing literature by exploring: (RQ1) how reports of HCI issues fit OSS designers' activities, (RQ2) what the information needs of designers in OSS projects are, and (RQ3) how to support users on the task of creating reports of HCI issues that meet those needs.

4 http://www.gnome.org

In order to address questions (RQ1) and (RQ2), we conducted interviews with four designers contributing to OSS projects, and qualitatively evaluated a set of 547 bugs reported under HCI-related keywords. To address question (RQ3), we designed a form for user reporting of HCI issues, based on the data collected in the aforementioned study, and on Semiotic Engineering. To investigate how well this form communicates the information needs of OSS designers, we performed a study in which participants were invited to report HCI issues through the designed form.

1.1 Document Structure

This dissertation is organized as follows: In chapter 2, we present the research questions we are targeting with this study. In chapter 3, we compare our work to previous research in five areas: OSS projects and HCI activities, end user reporting, bug reporting, formats for reporting HCI issues, and information needs in software development. In chapter 4, we present concepts and techniques explored in this work. In chapter 5, we present and provide a rationale for the research methods employed for this work. In chapter 6, we report our findings on how reports of HCI issues fit into OSS designers' activities and what the information needs of OSS designers are. In chapter 7, we describe the design and evaluation of a form for reporting HCI issues. Finally, in chapter 8, we wrap up this work with our conclusions and discuss opportunities for future work.

2 Research Goals

This chapter aims to summarize the background of the work presented in this dissertation, and to then formally state our research questions.

2.1 Background

Several authors (Cox, 1998; Crowston *et al.*, 2004; Gacek and Arief, 2004; Gallivan, 2008) have described OSS communities as having a hierarchical or onion-like structure, with increasingly larger groups of less technically savvy contributors. At the core, there is a small team of developers who contributes regularly and oversees the design and evolution of the project. Surrounding the core, there are the co-developers, people who contribute less often and by reviewing code or submitting bug fixes. Beyond this, come the active users, a technically savvy subset of the user base who contributes with bug reports and feature requests, but who does not actually code. Even farther from the core, there are the passive users, a group whose size is difficult to tell given the nature of OSS distribution channels (Crowston *et al.*, 2004). Passive users may include both technically savvy and non-savvy users.

When the open source movement started, developers involved in OSS

projects basically used to build tools for themselves to use. These tools were shared with other developers, who, by taking interest in a certain piece of software, could then come and help build it. With the proliferation of OSS, however, OSS communities now perceive a change on their user bases, which in the past was mostly composed by developers building for themselves. Anyone with an Internet connection can now download an OSS, whether or not this software was developed with that user demographic in mind. Bach and Carroll (2009) say "... a great number of [open source] software applications is available on the Internet for download, but the kind of experience that comes with that software is unpredictable".

In order for OSS projects to achieve wider adoption, especially in less technical user bases, we need to overcome the HCI problems that arise from having mostly technical people creating technical artifacts. As mentioned before, Nichols and Twidale (2003) outlined nine barriers in OSS projects that negatively impact HCI practices, among which reside issues related to the developers' perception of users and of their issues while using software (Table 1). Unsurprisingly, when proposing potential approaches to overcoming these barriers and improving HCI insertion in OSS projects, Nichols and colleagues (2006) point to end user involvement as a promising way to bridge the gap between developers and users.

Table 1: Barriers to HCI activities in OSS development adapted from Nichols and Twidale (2003).

Developer perception	Developers perceive users as equally technical or annoyingly stupid. Developers perceive usability problems as trivial. Developers perceive usability problems as functionality problems. Developers value power over simplicity: complex software is more powerful, but can be difficult to use.
Community integration	Difficult for usability professionals to integrate into open source culture. No resources for high quality usability work.
Process constraints	Difficult to innovate because of mental models already established from closed source software. Function-centric software that anybody can work on can result in 'feature bloat'.

As Pemberton (2004) observes, if OSS is to appeal to people other than the ones producing it, OSS communities need to (a) start learning what other people's itches are – in reference to Raymond's (1998) archetype of developers "scratching their personal itches" in OSS projects – or (b) empower people with ways to "tickle the programmers, so that they will scratch it."

2.2 Research Questions

This dissertation explores how user reporting of HCI issues can be used as a way to achieve the second approach proposed by Pemberton. Our goal is to support the involvement of the passive user layer of the onion-like structure of

OSS projects in order to provide OSS designers with data of people's actual use of OSS. We expect that this kind of data will not only support OSS designers in the task of redesigning OSS release after release, but also in the task of informing the rationale of design decisions to their respective OSS communities – a point we observed as very problematic, from our involvement in the GNOME project. We aim to employ user reporting as an alternative to traditional HCI evaluation methods, since those have proved themselves unsuitable to the OSS ecosystem (Andreasen *et al.*, 2007; Bach and Carroll, 2009).

In this work, we try to understand the information needs of designers in OSS projects and how reports of HCI issues fit into their activities. Based on this understanding, we then explore ways in which we can provide support for users to produce reports of HCI issues that align with these needs. Given this scenario, the following research questions apply:

RQ1 How do reports of HCI issues fit OSS designers' activities?

RQ2 What are the kinds of information OSS designers need in reports of HCI issues?

RQ3 How can we support end users in creating reports of HCI issues that meet OSS designers' information needs?

3 Related Work

The phenomenon of OSS development and communities attracted many research disciplines throughout the years, including business, information systems, computer-supported cooperative work, and software engineering, to name a few. We focus this overview of the literature on research that examines issue reporting practices and systems, from the perspective of Open Source Communities, Human-Computer Interaction and Software Development.

3.1 OSS Projects and HCI activities

Typically, OSS projects are organized around technically talented developers, whose communication revolves around technical aspects and source code. Although OSS communities generally acknowledge that a greater emphasis on HCI is essential for OSS wider adoption (Andreasen *et al*., 2006; Duffy, 2010), a culture of design and attention to HCI practices has not been very strong in OSS projects (Schwartz and Gunn, 2009). Even though OSS projects have managed to attract a large number of developers and users, Bach and colleagues (2009) observe that HCI designers still rarely engage in them, and that HCI activities are often neglected. They address motivation as a main challenge for

getting designers to participate in OSS projects, and propose fostering ways to provide opportunities for merit from HCI activities as a way to encourage designer participation in OSS communities.

Nichols and colleagues (2001) suggest that HCI activities in OSS projects may be restricted by the software development process itself. They described the results of a usability test of the open source Greenstone Digital Library software, and tried to understand the likely causes for the problems identified from within the Greenstone development environment. Among other problems, the lack of average (less technically savvy) user involvement was pointed out as one of the main sources of HCI issues. Indeed, this also figures in Nichols and Twidale's (2003) list of nine barriers in OSS projects that negatively impact aspects of HCI (Table 1), as a matter of developer perception.

Other researches described additional barriers for adopting HCI practices in OSS development. Hedberg and colleagues (2007) reviewed the literature on HCI from the perspective of OSS development. They captured a number of issues related to the nature of the process itself: developers use software differently than non-technical users; lack of a user-centered approach to software development; lack of designer participation; unsuitability of traditional HCI methods within the spirit of open source, among others. Viorres and colleagues (2007) reinforce these results by highlighting the problems of HCI methods integration into OSS development, and by explaining that HCI activities add another layer of complexity to the OSS development process. Validating the

results of these analytical studies, empirical studies confirm the challenge of integrating HCI methods into OSS projects (Twidale and Nichols, 2005; Andreasen *et al.*, 2006).

These barriers for HCI methods adoption, of course, are also perceived in terms of evaluation of OSS user interfaces. According to Andreasen and colleagues (2006), common sense is the primary evaluation method used in OSS development. Their study consisted of a questionnaire survey and a series of interviews with OSS contributors with both technical and human factors background. They found out that common usability conventions and guidelines frequently replace the use of traditional HCI evaluation methods. They also observe that, nevertheless, OSS designers firmly state that these tools are not sufficient for evaluation purposes, and that more formal user studies should also take place within OSS development.

3.2 User reporting

User reporting of issues is one of the most common approaches to post-deployment research, a subset of the remote HCI research methods. Hartson and colleagues (1996) define remote research methods as methods "wherein the evaluator, performing observation and analysis, is separated in space and/or time from the user." They present remote methods as an alternative to the traditional HCI research methods, where users are directly observed by evaluators, usually in a laboratory environment. In this work, they describe a list of possible approaches to remote methods for the purposes of formative

evaluation, that being “evaluation used for improving user interaction with systems.” Among the possible approaches figure remote questionnaires and surveys, videoconferencing as an extension of the laboratory, and instrumentation of an application and its interface for logging of usage data.

A subset of the research on remote methods focuses on the matter of post-deployment scenarios. While upfront user research and prototyping are crucial to the design of user-centered applications, learning about the actual ways in which users use an application after its deployment is also valuable (Norman and Draper, 1986; Nielsen, 1992). Hartson and Castillo (1998) state that “the need for usability improvement does not end with deployment, and neither does the value of lab-based evaluation, although it does remain limited to tasks that developers believe to represent real usage.” So far, however, the literature presented little work on HCI research methods for the post-deployment phase.

From the observation of the lack of research on the area, Chilana and colleagues (2011) surveyed 333 full-time usability professionals in large and small corporations from a variety of industries to better understand and characterize the state of post-deployment HCI activities. One of their findings suggests that, as observed by Nielsen (1992), the role of HCI experts tend to diminish after deployment and that they are rarely involved in post-deployment activities. They also note that, when HCI experts have the opportunity to be involved in post-deployment activities, their contribution is often “found of significant value.”

Nichols and colleagues (2003) point out two main approaches to post-deployment HCI research. In the first one, users' actions can be automatically recorded and sent to developers, through an instrumentation of the application and its interface. Hartson and colleagues (1996) state that this approach, to which they refer as instrumented remote evaluation, has the advantage of not interfering with the users' activities. They also observe, however, that it might be hard to infer interaction problems effectively from data gathered using this method.

The second approach pointed out by Nichols and colleagues is to allow users to proactively send reports on HCI issues (Castillo *et al.*, 1998; Hartson and Castillo, 1998). Hartson and colleagues (1996) stress the potential of this method, to which they refer as semi-instrumented remote evaluation, but also alert on the issues of relying on users with close to no training to identify interaction issues. Fortunately, Hartson and Castillo (1998) show that users, with brief training, are able to identify, report and rate the severity level of critical HCI issues. This claim comes from a study that compared evaluation of data obtained through laboratory observation and through user generated reports. With this study, Hartson and Castillo also observe that the technique simplifies data collection and reduce potential biases of an evaluator on the users and on the observations.

Nichols and colleagues (2003) present the only work we found that makes explicit the suitability of user reporting of HCI issues to OSS projects. In this

work, they also present a prototype for the open source Greenstone Digital Library software. They mostly use this prototype as a test case for the challenge of designing user reporting tools that fit their highlighted requirements, and present no evaluation of it. While evaluating remote HCI methods, Andreasen and colleagues (2007) also drew a line connecting remote methods in general and OSS projects: “OSS development is characterized by distributed collaboration between contributors to a specific project. A project can have hundreds of contributors spread worldwide. This makes it hard to employ conventional usability testing methods”.

Bach and Twidale (2010) also shortly explore the matter of how users with less technical skills can report on HCI issues, but focus mostly on the use of Schön's model of the reflective practitioner (1983) to engage these users in HCI discussions from a participatory design perspective. They analyzed the contents of HCI discussions within the OpenOffice project focusing on in-depth analysis of the participation of 5 users, and illustrated each of the three characteristics of Schön's reflective practitioner (problem framing, hypothesizing, and understanding) within it.

This work aims to explore the use of user reporting of incidents in the context of OSS, having in mind the advantages of post-deployment methods and its suitability to OSS development, exposed by the aforementioned works.

3.3 Bug Reporting

While there has been a number of studies on the dynamics of OSS projects,

little work has focused on the issue of bug reporting. Several studies make use of bug databases as a data point for observing other aspects of OSS development, such as coordination of work and community organization. There are some works on how to automatically assign developers to bug reports (Anvik *et al.*, 2006; Canfora and Cerulo, 2006), to track software features over time (Fischer *et al.*, 2003), to recognize bug duplicates (Čubranić, 2004; Runeson *et al.*, 2007) and to predict development effort for a certain bug report (Weiss *et al.*, 2007).

Prior work has shown that users of OSS do contribute with reports (Bettenburg *et al.*, 2008; Mockus *et al.*, 2002), confirming the existence of the active users group in the onion-like structure of OSS communities. From this starting point, Ko and Chilana (2010) investigated to what extent power users provide valuable contributions through bug reports. Their study consisted of an automated analysis of a data set of 496,766 bug reports collected from the Mozilla bug database. Results suggest that the value of bug reporting is in "finding talented reporters, instead of in deriving value from the masses".

According to Bettenburg and colleagues (2008), most work on what makes a good bug report constitutes of "anecdotal evidence", mentioning, for instance, the numerous articles and guidelines on effective bug reporting that can be found in the Internet. They conducted a survey among developers and bug reporters in the Apache, Eclipse and Mozilla communities in order to characterize what makes a good bug report. Their results show that there is a mismatch between what information developers consider to be important

and/or useful and what bug reporters usually provide in their reports. They suggest that the lack of support in bug reporting tools is largely responsible for this mismatch and, as an alternative, developed Cuezilla, a tool that measures the quality of a bug report and recommends additions to increase its quality.

Breu and colleagues (2010) quantitatively and qualitatively analyzed the questions asked in a sample of 600 bug reports collected from the Mozilla and Eclipse bug databases. They categorized the questions found in these reports (and in their follow-up comments) and analyzed developer response rates and times by both category and project. They note that reporters' ongoing participation is important for making progress on the resolution of the bugs they reported, since many follow-up questions are posed in order to achieve better understanding of the problem. Once again, the need for better bug reporting tools is brought to attention, given the necessity of better ways to elicit the necessary information from users.

The LibreOffice[5] community, for example, has recognized the difficulties non technically savvy users may face when using existing bug reporting systems. They elaborated a tool called Bug Submission Assistant[6] (BSA) with the purpose of replacing the regular Bugzilla reporting interface, which they claimed[7] to be more suited to “expert” users. The BSA reporting form consists of the required fields in the regular Bugzilla reporting form. These fields are

5 http://www.libreoffice.org

6 https://www.libreoffice.org/get-help/bug/

7 https://wiki.documentfoundation.org/Bug_Submission_Assistant

presented in a wizard-like manner (Figure 1), with the purpose of emphasizing to the user that every step in the process is "necessary to properly fill a bug report". As a replacement of Bugzilla, BSA mostly provides support for reporting software crash or missing features and has no particular emphasis on the reporting of HCI issues.

Figure 1: The Bug Submission Assistant wizard-like interface

Apart from the works of Bettenburg (2010) and colleagues and Breu and colleagues (2010), other researches also suggest the improvement of bug reporting tools as a way to increase developers' productivity (Just *et al.*, 2008; Breu *et al.*, 2010). Chilana and colleagues (2010), handle this matter from the

perspective of leveraging user participation in bug reporting. Their study on users' expressions of unwanted behaviors in bug reporting indicate the need for "more concrete ways (for users) to express a range of unwanted behaviors" in bug reporting tools.

There is little detail on the literature on the specific topic of bug reporting of HCI issues. Some of the works concerned with how bugs are processed within OSS projects (Scacchi, 2002; Crowston and Scozzi, 2004, Sandusky *et al.*, 2004a; Sandusky *et al.*, 2004b) touch on the matter of user interface (UI) bugs, but with no deep involvement in the topic. Twidale and Nichols (2005) examine bug reports collected from the Mozilla and GNOME Bugzilla bug databases, in order to characterize how designers address and resolve HCI issues. Their bug report data set consisted of bug reports described by keywords such as 'usability', 'human-computer interaction' and 'interface'. Some of their findings point to interface design problems with bug reporting tools. For example, they suggest that the text-centric nature of Bugzilla imposes challenges to the discussion of dynamic aspects of HCI.

Faarbord and Schwartz (2010) explored ways in which OSS communities can adapt their bug reporting tools in order to better capture usability issues for the purposes of evaluation. They proposed what they call a Distributed Heuristic Evaluation, which consists of associating bugs to what usability heuristic (from a set of pre-defined usability heuristics) they violated. As far as we know, the technique was not evaluated in any sense. However, Farboorg and

Schwartz mention that they expect that the use of the distributed Heuristic Evaluation intra and inter OSS communities would build a shared vocabulary for describing HCI issues.

There are other works on vocabulary matters when it comes to users reporting issues they experience while using a certain piece of software. Chilana and colleagues (2010) conducted a study on how end users express wanted behaviors in bug reporting. They created a classification of seven common expectation violations cited by end users in bug report descriptions and applied it to 1000 bug reports from the Mozilla project. Their findings show that reporters tend to describe bugs as violations of the community's expectations, instead of their own. Additionally, they also noted that bugs described as violations of personal expectation are less likely to get fixed.

Ko and colleagues (2006b) present a linguistic analysis of how people describe software problems in bug reports. They performed a quantitative study over nearly 200,000 bug report titles and discuss several design ideas for bug reporting tools, motivated by their study results. They suggest a redesign of bug reporting forms in order to structure the information reporters naturally include. For example, for the case of report titles, their study indicate most bugs are summarized by (1) a software entity, (2) a quality attribute, (3) a problem, (4) the execution context, and (5) whether the report is a bug or a missing feature.

Outside the context of OSS development, Heller and colleagues (2011)

describe a prototype for bug reporting focused on simple one-bit-feedback facilities, such as the Facebook “Like” button. The prototype consists of a physical hardware button, which users are supposed to press whenever they observe an incident. Users may provide further details if they want, but this is not solicited to them given the occurrence of an issue. No studies on the usefulness of these one-bit reports on the eyes of designers were presented.

In conclusion, bug reporting is currently one of the main ways users contribute to OSS projects with both code and HCI issue reports. The works presented in this section, however, highlight the unsuitability of current bug tracking tools to the report of HCI issues, and the need for better tools to support users in creating reports that are actually feasible to OSS designers to work on, a gap we aim to explore with this work.

3.4 Formats for reporting HCI issues

We found four different formats in the literature (Jeffries, 1994; Mack and Montaniz, 1994; John and Packer, 1995; Lavery *et al.*, 1997) for reporting HCI issues. These formats are intended to be used by designers reporting findings obtained through traditional evaluation methods - such as the heuristic evaluation, the cognitive walkthrough, among others.

Based on the analysis of a collection of usability problem reports, Jeffries (1994) suggested a format that includes a description of the problem observed along with its severity, and a solution to it. She states that the description of the problem should mostly focus on the user and on the task he is attempting to

conclude, which results in causes and consequences of a given problem being discussed as a single topic. Lavery and colleagues (1997) argued against this format based on Hollnagel's (1993) work, which stresses the importance of distinguishing observed difficulties from inferred causes.

Mack and Montaniz (1994) based their format on reports developed as the diagnosis of inspection and walkthrough methods. They suggest that usability problem reports should include the type of the problem being faced, the task trying to be achieve, and the action associated with the problem's occurrence. A textual open-ended description should also be included, even though it is the more discrete information that constitutes the core of the report, since their main goal was to improve analyst effectiveness.

John and Packer (1995) propose a usability problem format to be used as output for the Cognitive Walkthrough method. Similarly to Jeffries's (1994) work, they suggest that a usability problem should be described by a statement of the problem and of its severity. It also asks designers to identify the frequency with which they run into the problem, and "an assessment of whether these judgments came from the technique [Cognitive Walkthrough] itself (....) or from some form of personal judgment." This format, as the aforementioned ones, also fails to highlight distinct aspects of a given usability problem, such as causes, consequences, and others.

As part of a broader study, Lavery and colleagues (1997) tried to address the matter by listing four components for any observed usability problem: "a

cause, a possible breakdown in the user's interaction, and an outcome, all of which happen in a context." Based on this understanding of what constitutes a usability problem, they proposed the usability problem report form presented in Table 2. Lavery and colleagues' approach is more focused on "validation for design research purposes", rather than on effectiveness and productivity, which was a concern for the aforementioned works.

Table 2: Usability problem report form proposed by Lavery and colleagues (1997).

component	**question**
Context	Please describe the context in which the problem arises
Cause	What is the cause? What is the type of the cause (e.g. Knowledge requirement, design fault)?
Breakdown in user's interaction	Did a breakdown of the user's interaction occur (Yes or No) -- if you answered "no" to the above question, go to component *Outcome*. What was the breakdown suffered?
Outcomes arising from breakdown	What was the user's behavior following the breakdown? What was the effect on the user's performance and work? -- go to component *Solution*.
Outcome	Did the cause change the user's behavior? How? What was the effect on the user's performance and work?
Solution	What is your recommended solution to this problem?

The formats described here are intended to be used by designers as a way to document HCI issues observed through evaluation methods. This dissertation contributes to the existing literature by providing a format to be

used by users, as way to report HCI issues they experience when using software.

3.5 Information needs in software development

Apart from the research of Breu and colleagues (2010) mentioned in section 3.3, other works examine the information needs of developers in various contexts. Ko and colleagues (2007) observed 17 developers at a large software company to understand the types of information developers sought and the sources they used to find information. From this, they derived a set of 21 types of information along with their outcomes and sources. They noted that, very often, developers had to defer tasks because the only source for a certain information was an unavailable coworker. Herbsleb and Kuwana (1993) investigated the information needs of software analysts. They analyzed the content of real software design meetings in three distinct organizations, focusing on the questions software analysts asked each other. Their findings suggest that most questions in their sample of software design meetings concerned software requirements and user scenarios, and that rationale for software design decisions are seldom asked.

Sillito and colleagues (2006) conducted two studies based on the observation of developers performing change tasks to medium to large sized programs, aiming to understand the types of information a developer needs about a code base in order to be able to perform changes to it. Their main contribution was a set of 44 categories of questions asked by their participants.

With the purpose of exploring how developers gain understanding about an unfamiliar code base, a similar study was conducted by Ko and colleagues (2006a), during which developers were asked to work on debugging and enhancement tasks for 70 minutes. Their results mostly focus on patterns on how developers browse code and on the problems they face while doing that.

We are not aware of any works on specific matter of the information needs of designers in software development. This work aims to contribute to the literature by addressing this gap, specifically for the case of OSS development.

3.6 Summary

Considering the works presented in this chapter, we are not aware of studies that explore how reports of HCI issues fit into OSS designers' activities. This study aims to contribute to the existing literature by addressing this gap, and investigating RQ1) how reports of HCI issues fit OSS designers' activities, RQ2) what the information needs of designers in OSS projects are, and RQ3) how to support users to produce reports on HCI issues that align with those needs.

4 Concepts and Techniques

In this chapter, we outline and describe the concepts and techniques that are important for the understanding of the work presented in this dissertation. We contextualize these concepts and techniques further in the chapters 5, 6 and 7, where we describe their use for the purposes of data analysis and theoretical foundation.

4.1 Heuristic Evaluation

Heuristic Evaluation (Nielsen and Molich, 1990; Nielsen, 1994a) is a Usability Engineering (Nielsen and Hackos, 1993) method for finding usability problems in a system through inspection of its UI. The application of the Heuristic Evaluation comprises having evaluators examine the UI and judge it based on its compliance with a set of recognized usability heuristics. Nielsen and Molich (Nielsen and Molich, 1990; Molich and Nielsen, 1990) proposed an initial set of usability heuristics to be used for Heuristic Evaluation. In 1994, Nielsen (1994b) revised those heuristics based on an analysis of 249 usability problems. The revised set of heuristics is presented in Table 3.

Table 3: 10 Heuristics for UI Design proposed by Nielsen (1994b)

Visibility of system status	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
Match between system and the real world	The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
User control and freedom	Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
Consistency and standards	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
Error prevention	Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
Recognition rather than recall	Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
Flexibility and efficiency of use	Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
Aesthetic and minimalist design	Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
Help and documentation	Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

The output of a Heuristic Evaluation is a list of the identified usability problems, along with references to which usability heuristics were violated in each case. Nielsen (1995a) advised that evaluators "should try to be as specific as possible and should list each usability problem separately". He mentioned that if, for example, an evaluator identifies three problems with a certain UI element, all three problems should be listed in the evaluation report, with the specific references to the usability heuristics that explain why each constitutes a usability problem.

Nielsen (1995b) stated that "the lists of usability problems found by Heuristic Evaluation will tend to be dominated by minor problems". Jeffries and colleagues (1991) observed that the results obtained through the application of the method tend to report a large number of "specific, one-time, and low-priority" problems. Jeffries and Desurvire (1992) mentioned that the Heuristic Evaluation appears to expose a smaller amount of more severe, more recurring, and more global problems when compared with laboratory usability tests.

4.2 Semiotic Engineering

Semiotic Engineering (de Souza, 2005a) is an HCI theory centered in communication, characterizing HCI as a particular case of human communication mediated by computational systems. Semiotic Engineering views a system's interface as a designer-to-user message that represents the designer's solution to what he believes to be the users' problems, needs and preferences (de Souza, 2005a; de Souza, 2005b; de Souza and Leitão, 2009). The contents of this message, the metacommunication message, can be paraphrased as follows:

> "Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision" (de Souza, 2005a).

Through the use of static, dynamic, and metacommunication signs, which can be presented in various forms, such as words, behaviors and graphics, the system's UI becomes a representation of its designers at interaction time, becoming what is referred in Semiotic Engineering as the designer's deputy. According to de Souza (2012), "the interface enables all and only the designed types of user-system conversations encoded in the underlying computer programs at development time. The metacommunication message from the designers is unfolded and received as users interact with it and learn 'what the system means'."

The users' response to the designers' metacommunication is also mediated

by interface signs. In order to express their communicative intent, users must understand or learn the signification system in which the designers' message is encoded. This means that, when using a system, users may consistently fail to express their intentions because the system designer has not “anticipated the users' sign-making strategies” (de Souza, 2012). It is worth observing, however, that Semiotic Engineering's main investigation object is communication, not learning, of the operational, tactical and strategic aspects of metacommunication. Designers, from the perspective of Semiotic Engineering, should be concerned with not only producing a system, but also with introducing it to users (de Souza, 2005b).

4.2.1 Communicability Evaluation Method

Communicability refers to the system’s ability to communicate the designers’ communicative intent. One of the main evaluation methods in Semiotic Engineering is the Communicability Evaluation Method (CEM), which evaluates the communicability of computer systems as experienced by users – that is, the quality of the reception of the designer's message (de Souza *et al.*, 1999). The method consists of identifying the communicative breakdowns that occur while a user interacts with a computer system, and of the classification of these breakdowns using a set of predefined user utterances (Table 4). A communicative breakdown is an indication that the designer has failed to convey his message through the user interface. The utterances are used then to characterize the user's reaction when a communicative breakdown occurs.

4

Table 5: Description of the user utterances used in CEM (de Souza and Leitão, 2009).

user utterance	**illustrative symptoms**
I give up	The user believes that he cannot achieve his goal and interrupts interaction.
Looks fine to me	The user believes he has achieved his goal, although he has not.
Thanks, but no, thanks	The user deliberately chooses to communicate his intent with unexpected signs, although he has understood what preferential designer's solutions are promoted.
I can do otherwise	The user communicates his intent with unexpected signs because he cannot see or understand what the system is telling him about better solutions to achieve his goal.
Where is it?	The user knows what he is trying to do but cannot find an interlace element that will tell the system to do it. He browses menus, opens and closes dialog boxes, etc., looking for the particular sign.
What happened?	The user does not understand the system response to what he told it to do. Often, he repeats the operation whose effect is absent or not perceived.
What now?	The user does not know what to do next. He wanders around the interface looking for clues to restore productive communication with the system. He inspects menus, dialog boxes, etc., without knowing exactly what he wants to find or do. The evaluator should confirm if the user knew what he was searching (*Where is it?*), or not (*What now?*).

user utterance	**illustrative symptoms**
I give up	The user believes that he cannot achieve his goal and interrupts interaction.
Looks fine to me	The user believes he has achieved his goal, although he has not.
Thanks, but no, thanks	The user deliberately chooses to communicate his intent with unexpected signs, although he has understood what preferential designer's solutions are promoted.
Where am I?	The user is telling things to the system that would be appropriate in another context of communication. He may try to select objects that are not active or to interact with signs that are output only.
Oops!	The user makes an instant mistake but immediately corrects it. The “Undo” operation is a typical example of this user utterance.
I can't do it this way	The user is involved in a long sequence of operations, but suddenly realizes that this is not the right one. Thus, he abandons that sequence and tries another one. This user utterance involves a long sequence of actions, while *Oops!* characterizes a single action.
What's this?	The user does not understand an interface sign and looks for clarification by reading a tool tip or by examining the behavior of a sign.
Help!	The user explicitly asks for help by accessing “online help”, searching system documentations, or even by calling the evaluator as a “personal helper”.
Why doesn't it?	The user insists on repeating an operation that does not produce the expected effects. He perceives that the effects are not produced, but he strongly believes that what he is doing should be the right thing to do. In fact, he does not understand why the interaction is not right.

4.3 Summary

In this chapter, we presented the concepts and techniques that are relevant to the understanding of the research presented in this dissertation.

In this work, we used both Nielsen's heuristics and the utterances of the CEM as coding templates for the bugs reports evaluated as part of our investigation (section 5.2). Additionally, based on Semiotic Engineering, we designed a form through each users can report HCI issues (chapter 7). We describe the employment of these concepts and techniques in the chapters 5, 6 and 7, where we describe their use for the purposes of data analysis and theoretical foundation.

5 Methodology

In order to address RQ1 and RQ2 presented in section 2.2, this study investigates how reports of HCI issues fit into OSS designers' activities and identifies the types of information that designers in OSS projects need in order to understand HCI issues a given user – possibly technically naïve – faces when using a certain piece of OSS.

We performed two studies to articulate our findings. One study consisted of interviews with designers working on two major OSS projects, the GNOME project, a desktop environment for computers running GNU/Linux, and the Fedora[8] project, a GNU/Linux distribution. The other study consisted of an analysis of bugs reported in the Bugzilla instance of the GNOME project and tagged under HCI-related keywords. In this chapter, we present the goals established and the methodology used for each of these studies.

5.1 Interviews with OSS designers

In order to address both RQ1 and RQ2, we conducted interviews with OSS designers to understand how evaluation, feedback seeking and review, and

8 http://fedoraproject.org/

redesign activities take place within their work routine as contributors to the OSS projects they work on.

To carry out this study, we needed to interview people with previous experience working as HCI designers within the context of OSS projects. We conducted five semi-structured interviews (Appendix A) with participants who were recruited by sending invitations to the design teams of the Fedora and the GNOME projects. One of the interviews had to be discarded, since the participant's profile did not match the requirements for this study: his contributions as a designer to OSS projects were focused mostly on aesthetic aspects of the interface (such as icon design), and not on its interaction aspects.

Since participants lived abroad, the interviews were conducted via video chat, using the Google Hangouts video chat tool. Due to hardware limitations, we recorded only the audio of each interview, later transcribed by the researcher conducting the study, all with the participants' consent.

The interview transcripts were analyzed twice, each time with a specific research question in mind to guide the analysis. To address RQ1, “how do reports of HCI issues fit OSS designers' activities?”, we employed a variation of the Thematic Analysis technique, called Template Analysis (King, 2012). The central idea of this technique is to develop an initial coding template, which will evolve iteratively with the analysis of the collected data, allowing the definition of additional themes as needed. In our case, considering the question guiding this analysis, we selected three initial themes: feedback seeking, feedback

obtainment, and feedback usage. This initial set of themes was refined and updated multiple times after successive readings of the transcripts. We considered our template final when it became descriptive enough, without becoming overly detailed.

To address RQ2, "what are the kinds of information OSS designers need in reports of HCI issues?", we employed Thematic Analysis (Braun and Clarke, 2006), a qualitative method for identifying recurrent patterns (themes), to analyze our data. While transcribing the audio from our interviews, we started analyzing the data creating themes that described kinds of information in user feedback that OSS designers find useful and needed for their activities.

The final coding systems for each of our analysis of the transcripts (themes and template, respectively) is presented in chapter 6, along with the findings of this study.

5.2 Analysis of Bug Reports

Still addressing RQ1, we examined a subset of the bugs reported under the Bugzilla instance[9] of the GNOME project. Our purpose with this examination was to gain a sense of the scope of HCI problems currently reported in an OSS project. We tried to identify common characteristics in our bug data set, focusing not only on the nature of the reported problems, but also on how they are reported and by whom. The results of this analysis, in this study, were used to validate the findings from our interviews with OSS designers through

9 https://bugzilla.gnome.org/

triangulation.

We manually selected and analyzed a total of 547 bug reports from seven different products, which are smaller projects inside the GNOME community, and reported under two main keywords. In Bugzilla, keywords are predefined by the Bugzilla instance administrator and can be used to tag and categorize bugs within the bug database. Among the keywords registered for the GNOME's instance of Bugzilla,[10] we found three that matched our interest for this research, which was to evaluate reports related to HCI problems. Those keywords were `HIG`, `ui-review` and `usability`, whose descriptions, according to GNOME's instance of Bugzilla, are presented as follows:

`HIG`	Bug reporting an area where an application does not follow the HIG (GNOME's Human Interface Guidelines).
`Ui-review`	Indicate a bug field as part of the ui-review. Should be used only by the ui-review team.
`usability`	This keyword described a usability/user interface change where the correct behavior is not necessarily obvious and input from the usability team is desired.

The ui-review tag is suggested by the GNOME Design Team[11] as a way to get in touch with GNOME designers and report design issues. During our inspection of GNOME's instance of Bugzilla, we observed that the keyword is indeed used by others than the people in the ui-review team – which, as

10 https://bugzilla.gnome.org/describekeywords.cgi

11 https://live.gnome.org/Design

informed by GNOME designers via the #gnome-design channel in irc.gnome.org, is the Design Team itself. Additionally, we chose to discard the `HIG` keyword, since it is mostly related to interface compliance to the GNOME Human Interface Guidelines,[12] rather than to problems faced by users.

Table 6: Description of products selected for bug analysis.[13]

empathy	Empathy is a messaging program which supports text, voice and video chat and file transfers over many different protocols. Empathy is the default chat client in GNOME, and is based on the Telepathy framework, making it easier for other GNOME applications to integrate collaboration functionality.
epiphany	Epiphany is the GNOME web browser based on the WebKit rendering engine.
evolution	Spiffy mail/calendar/address book/task list application.
gedit	gedit is a small lightweight text editor for GNOME.
gnome-control-center	Settings
gnome-shell	Next generation GNOME desktop shell
nautilus	The GNOME file manager.

GNOME's instance of Bugzilla contains bugs related to very different pieces of software, ranging from “bindings” and “low-level software”, to “core desktop functionality” and “applications”. We chose to focus on products whose

12 https://developer.gnome.org/hig-book/stable/

13 https://bugzilla.gnome.org/browse.cgi

audience we considered most likely to also include less technically savvy users. With this in mind, we decided to pick for analysis bugs associated with two pieces of desktop functionality (gnome-shell and gnome-control-center), and five standalone desktop applications (empathy, epiphany, evolution, gedit and nautilus). A description of the selected products, according to the information in the instance, can be seen in Table 5.

Finally, in a bug database with more than 700,000 bug reports, filtering by product and keyword still returned a quantity of reports that was unmanageable for a manual, qualitative investigation (the procedure is described later in this section). Because of this, we decided to apply a creation date filter to our data set, filtering out bugs created before January 1st, 2010. In the case of the pairs {product: gnome-control-center, keyword: ui-review} and {product: gnome-shell, keyword: ui-review}, the quantity of bug reports was still unmanageable within our time frame, so, for these products, we filtered out bugs created before June 1st, 2011.

Bug reports that matched the aforementioned filtering criteria (Table 6) were investigated in more depth. Among the many fields in a bug report, we observed reporter (user or developer, according to how they were labeled by Bugzilla), description, and attachments. We focused this analysis on the initial reporting of a bug, not investigating in depth the subsequent discussion about the problem it reports and its possible solutions.

We used the “description” field as our main source of investigation. We

analyzed its contents from a qualitative perspective, following the more ethnographic low-level style of investigation presented by Twidale and Nichols (2005) in their study of usability discussions in OSS development. This means that, when reading reports, we tried to focus on what was “in some sense 'surprising' in the light of (...) standard HCI research” (Twidale and Nichols, 2005).

Table 7: Our bug sampling per product and per keyword investigated.[14]

product	keyword		Total
	ui-review	usability	
empathy	5	10	15
epiphany	6	17	23
evolution	5	45	50
gedit	0	14	14
gnome-control-center	93	24	117
gnome-shell	236	39	275
nautilus	14	39	53
			547

In addition, to gain a better understanding of the kind of HCI problems reported, we also tagged the description of each bug using two different sets of tags, one based on the heuristics of the Heuristic Evaluation (Nielsen, 1994b; section 4.1), and another based on the user utterances of CEM (de Souza *et al.*, 1999; section 4.2.1). The elaboration of this procedure was based on the work of

14 https://bugzilla.gnome.org/browse.cgi

Salgado and colleagues (2006), a comparative study of three distinct HCI evaluation methods: Cognitive Walkthrough, Heuristic Evaluation, and CEM. In our work, we mostly focus on the differences observed between the last two methods, especially in terms of the kinds of problems identified by each of them. Examples of the procedure can be seen in Appendix B.

According to Salgado and colleagues, the problems observed by Heuristic Evaluation are mostly related to *interface* characteristics, noting the presence or absence of general design guidelines. CEM, on the other hand, proved itself to be more suitable to the identification of *interaction* problems, revealing issues related not only to the execution of a specific procedure, but also to the logic applied to it. With that in mind, we believe our analysis of the reports in the light of those two evaluation methods highlighted tendencies in their contents in terms of the distinction of interface and interaction issues.

The findings of this analysis of bug reports is detailed in chapter 6.

6 Feedback Management and Information Needs

In this chapter, we present the results of the studies described in chapter 5, and discuss our findings with regards to RQ1 and RQ2 (section 2.2). The four interviewees that participated in our interviews will be identified in this chapter as P1 to P4.

6.1 RQ1: How do reports of HCI issues fit OSS designers' activities?

As mentioned in section 5.1, the transcripts from the interviews with OSS designers were analyzed twice, each time with a specific research question in mind to guide the analysis. The first analysis, conducted using the Template Analysis technique, aimed to explore RQ1, “how do reports of HCI issues fit OSS designers' activities?” The coding template originated from the Template Analysis of the interviews consists of a hierarchy of themes, in which the first-level themes encapsulate the second-level themes. The result is the coding template shown in Table 10.

In this section, we describe each of the first-level themes in function of the second-level themes related to it. Each theme reveals aspects of how feedback

and reports fit OSS designers' routine and activities. When pertinent, we triangulate those findings with the findings of our analysis of bug reports. We have omitted the identification number of the bug reports we quote, as well as their authors. However, all the information on the bug reports we evaluated is publicly available online.

Table 8: Coding template created after the interviews from the first study (section 5.1)

First-level themes	Description	Second-level themes	Description
A) Designers' activities	What activities OSS designers perform in their routine	A.1) Design activities	How OSS designers approach user interface design
		A.2) Evaluation activities	How OSS designers conduct evaluation of their design solutions
B) Getting feedback	How and why OSS designers seek and obtain feedback	B.1) Purpose	Why OSS designers want and need feedback
		B.2) Sources	Where OSS designers seek and obtain feedback
		B.3) Designer's attitude	Feedback-seeker and feedback-receiver attitudes
C) Using feedback	How OSS designers interact with obtained feedback	C.1) Motivations	Why OSS designers work on a certain piece of feedback
		C.2) Obstacles	Problems faced by OSS designers when managing feedback

A) Designers' activities

The themes under “Designers' activities” describe activities OSS designers tend to perform within OSS projects. These themes reveal aspects of how common design process stages take place within OSS projects.

A.1) Design activities

This theme is related to the activities the interviewees perform when designing or redesigning a user interface solution. All OSS designers interviewed mentioned **goal and requirements** definition as the first task in their design/redesign processes.

> "So, in the case where I did the [software name] application, to begin with, I had a problem described. I had a **specification of the objectives** of the application." (P1)

> "That [initial process] generally takes the form of identifying a need, and what are the **goals**... That is something we put a lot of emphasis on: being very clear about what kind of problems we need to solve for the user, and how it's going to fit the overall system." (P4)

Most of the interviewees stated that those requirements are often derived from evaluating **existing solutions**, and from what they consider to be standard knowledge on user interface design.

> "(...) I'd say we define those goals we want to address, and then we look at **relevant art**, we look at existing solutions and see what we like and don't like of each one (...)" (P4)

> "I guess [I derive requirements] from common sense, other systems, books, articles, past experience... (...) And also what **other pieces of software solve this particular problem**, and try to see what they did well and what they did not do so well." (P1)

Some interviewees also mentioned the **community as an input source for defining goals** and requirements of a certain feature or application.

> "I speak with people who are, hm, what I would call the **stakeholders** of a feature, when I start to design a new feature. A lot of times that would be somebody, hm, like, **community members**. (...)" (P3)

> "I suppose what is relatively unique in [*name of the project*] in terms of design might be... Generally speaking, **we are dependent on developers and volunteers coming forward and discussing those projects**." (P4)

Once the goals of the design are considered to be well-established, all OSS designers stated their next step in the process would be to start creating **wireframes and mockups** of the actual interface solution. Those materials are considered by the interviewees to be the **core output of their design activities**.

> "I usually work on sketches that I basically use as a personal way to get ideas in paper, but I usually don't use it for other people's consumption, if that makes sense. (...) **When I want to get feedback, I usually work on wireframes and mockups**, on Inkscape." (P3)

> "[What is the output of your design activities?] Mostly **wireframes and mockups**. They are often enough to give an idea of how the thing should work, and to open a dialogue with the developers of that thing, at the same time." (P1)

Some interviewees, however, mentioned they tend to **avoid committing to complete specifications** of their mockups and wireframes. They stated that they usually leave a lot of things on the design solutions to be decided and negotiated along with the developers who will implement the feature or application.

> "We mostly produce snippets of documentation, tops. The funny thing is that

> our sort of **documentation process happens mostly over a dialog with the person who's supposed to implement it**, rather than something that is written down. That makes some room for negotiation." (P1)

> "We usually take the process to a stage where we have initial designs, we have goals, but we usually have not worked out all the details. That's partly because **we're leaving quite a lot of fruits dangling around to figure out with developers**. We don't get a say on what developers should be working on, so we try to stick together with them." (P4)

> "I mean, we work with them [developers] to flesh out the details, we try to have design and development fairly coordinated. **We don't really try to have complete specifications** because it doesn't fit quite very well with the kind of process we have, where **we want developers quite involved in design**, helping to solve problems as we go." (P2)

P2 also illustrates how the **collaboration with developers** might also influence their process in terms of redesign:

> "(...) **they [developers] come to us and ask for a mockup or for explaining a certain mockup**. Like, "hey, we have this mockup, and we're having this kind of issue with it, can you change it?" or "what can we do?". You know, this kind of back and forth with developers." (P2)

As it is possible to observe, interaction with developers plays a big role in the design activities performed by the OSS designers interviewed. This means that communication and iteration with developers influence design decisions and directions.

A.2) Evaluation activities

This theme uncovers aspects of how evaluation takes places within the interviewees' activities as an OSS designer. All the interviewees mentioned **inspection methods**, such as the Heuristic Evaluation, as their main tool for evaluation. Often applied close to the end of release cycles, those methods often take the form of a non-systematic exploration of the user interface, trying to cover what are expected to be the main interaction paths of the software in question.

> "One thing that I tend to do a lot is to build the stack and explore it, trying to **identify simple heuristic UI issues**." (P1)

> "I take our unstable releases, and I run them, inspect them. Then I identify stuff that looks wrong. Like, if there's something like... You know, some regular **Nielsen Heuristic Evaluation** and usability studies." (P2)

> "When we develop a piece of software, we will tend to, at some point towards the end of the development process, do a review of some kind. That will generally be more of a **heuristic thing**, where we look at the software in detail." (P4)

The OSS designers interviewed also observed that the **outcome** of this kind of evaluation usually takes the form of **a series of bug reports** that should ideally be fixed before the end of the release cycle.

> "So then [after building and reviewing the software], I tend to screen shot the problems and **report them as issues in the bug tracker**". (P1)

> "At this point [when doing inspection close to the end of the release cycle], **I will be filing a *lot* of bugs**. That's kind of a polishing and refinement stage, when we

> are talking about a new piece of software. If it is something that is already out there, then it is more of an ongoing process of release review." (P4)

Once a software is released, evaluation activities also take the form of what P2 called "**Bug Review**". This basically means browsing through bugs reported under certain reserved keywords (such as "usability" or "ui-review"), and then reviewing the design as/if needed. P2 explains it as follows:

> "And there are things that we do all that time, which is doing **bug review**, for example. Looking through the bugs that people filed against it [the software] under the tag of usability, and making sure we didn't miss anything, if it's something we did talked about before or if it should be included in the mockups." (P2)

All the OSS designers interviewed mentioned Bug Review as a **daily activity** and one of the most **time-consuming** ones.

> "**Reviewing bugs usually takes a lot of time** because we need to understand what this reporter meant, and, most of the time, engage with them in order to get missing information. Now imagine doing that for fifteen bugs a **day**." (P1)

> "I have to engage in some sort of back and forth with reporters about 80% of the time, just to figure out what they were trying to do, why, and in which ways. **I invest a big part of my time on this**. Much more than I'd like to." (P2)

Usage of **investigation and observation methods is not usual**, according to the interviewees. P1 and P3 observed **user recruitment and participation** as the main constraint for the use of methods from this nature. P4 agrees with

them, but also adds that investigation and observation methods are often too **time-consuming**, making their usage unmanageable within short release cycles and small design teams, both characteristic of OSS projects. Among the OSS designers interviewed, only P2 had applied observation methods in the context of OSS development. P2 recognized the problems stated by the other participants and also added **communicating the value of this kind of method to developers** is not often easy:

> "I guess the problem, the reason that we don't do it [user testing] as often as maybe we should is that **you really need the development team to commit to it**. You know what I mean, you don't want to spend, like, 3 months with usability tests and getting all the data there, to get developers, like, "whatever!". You want them to commit and be sure they are gonna care and get the community to care." (P2)

Apart from that, some of the interviewees also noted that, given the collaborative nature of OSS development, **reflection and discussion on the designs** is a constant activity during the development process. P4 illustrates this when asked how evaluation takes place before the actual release of software:

> "We are always reflecting on what we produce. There is always people coming forth with different perspectives and thinking quite deeply about the different use cases. So, hm, somehow **we evaluate the design as it goes.**" (P4)

Concluding, from what was exposed by this theme, it is possible to observe that bug reports play an important role on the evaluation activities

performed in OSS projects. Depending on the source, they might act as input for evaluation, or as outcome of it.

B) Getting feedback

The themes under "Getting feedback" uncovers aspects of why and how OSS designers deal with feedback from users.

B.1) Purpose

This theme addresses the different reasons why OSS designers value and consider feedback from users as an important input for their design activities. All the OSS designers interviewed claimed to use feedback from users as **input for evaluation activities**. They observed that feedback analysis often leads to either the redesign of a user interface (or, more frequently, of parts of it), or the "refinement" of a certain piece of the user experience. P1 and P3 summarize this, by saying:

> "Well, this [feedback] is the way I get review, hmm, **evaluate what I do**, what we create. We often get an overall feeling of how people are reacting to a release, but it's hard to tell what exactly is wrong. It's much valuable when we get to see what's actually the problem. I tend to keep this kind of stuff in the back of my mind when... I mean, for example, for the Shut Down button, we got a lot feedback from various people from the internet, we thought about it, then we changed it." (P1)

> "But it's [why I look for feedback] like... Are people responding positively to it? Or are they responding negatively? If they are responding negatively, is it because of a technical difficulty? Like, is something not working correctly? Or is it a bug? **Or is it because of actual poor design**? Or poor User Experience? This is the kind of thing you can use to iterate on a existing design and try to make it better, and, you know, start all over again." (P3)

P3 also mentioned that this kind of feedback-oriented evaluation is **more suitable to OSS development** than other kinds of evaluation methods. P3 explains this by pointing again the difficulties of applying other evaluation methods that require user participation:

> "In an ideal world you'd have some user testing, that you could put users in front of it [software], see how they do things and use that to judge the design. **I very rarely get that chance**, to have access to actual users. So, in a lot of ways, it kind of... You kind of have to judge by yourself and from the feedback that you get from developers and the people that use it." (P3)

In addition, three of the four interviewees mentioned feedback as a way to **leverage HCI activities** among OSS community members. In this context, feedback might be presented as a way to both inform design decisions and to reinforce the value of HCI design within the OSS development process. P1 and P3 illustrate these uses, as follows:

> "You know, when people provide good feedback, we are able to **better inform our decisions** as well. I mean, like... We can then say: "We changed that, because people had this and this problems with the previous solution". People don't deal very well with change. When we are able to explain changes based on real feedback, people seem less resistance. **They then believe on our work**." (P1)

> "A lot of problems come from half-implemented designs. Feedback on problems that occur because of half-implementations is often a great way to communicate the value of our work to developers. You know... "Hey, remember this was supposed to do this too?" or "We were supposed to have this other stuff too", you know? Then I get developers to get back to that and get work done, so we can provide fixes to people. Feedback is often how **I can inform developers why**

it is important to implement a feature or something." (P3)

It is possible to observe that user feedback is an important tool for designers in OSS projects. It enables them to iterate on their design decisions, explain them to the broader OSS community, and to influence the development process, based on users' needs.

B.2) Sources

This theme describes where OSS designers seek and obtain feedback from their users. All four interviewees also mentioned the **social media websites** Google+ and Twitter as an important source of feedback, even though it was observed that the **contents of this kind of feedback tend to vary**. P3 exemplifies this, by saying:

> "It really **depends on the source**. Bug reports are a bit more detailed. Social media... Like, Twitter, of course, you only have a 140 characters, right? So it's usually like "I couldn't do X, this sucks!", you know? On Google+, people tend to be a bit more descriptive, but the somewhat nasty tone is there." (P3)

P2 exposes the same issue, but mentioned that, when having the opportunity to **engage with the reporter** through those social media websites, he gets to figure out missing information and, then, file a more complete bug report:

> "Occasionally, people on Google+ or Twitter, whatever, complain about it [the design]. And if I have the time, when I see it, I'll **try to engage with them** and try to figure out what they were trying to do. It's actually more useful than a bug report, because, if the person engages with me, then I can try to find right

> away "oh, you were trying to do this". I do end up filing bugs that way, but having the whole context in mind then." (P2)

Blog posts are another relevant source of feedback mentioned by the OSS designers interviewed. This kind of feedback usually consists of a compilation of HCI issues, and/or an expression of how one feels about a certain OSS or OSS community. Blog posts might be **time-consuming** for OSS designers depending on their **repercussion** in the Comments section or in social media websites like reddit,[15] Hacker News[16] and Slashdot.[17] Feedback obtained through blog posts often generate more feedback.

Finally, three of the four interviewees mentioned that they frequently obtain **in-person feedback**. P2 and P4 mentioned that a lot of feedback comes from meeting people during OSS events, project-focused or not:

> "Or just by **meeting people** [I get feedback]! I was at a conference just last week and I think, well, pretty much everyone that I spoke to had a piece of feedback or opinion about the design. There's some useful stuff out there, but you gotta spend some energy on processing it." (P2)

All the OSS designers interviewed mentioned that feedback coming from what P4 called "**inaccessible users**" is very important and valuable. Considering the onion-like structure of OSS communities mentioned in section 3.1,

15 http://www.reddit.com/

16 https://news.ycombinator.com/

17 http://slashdot.org/

"inaccessible users" would be equivalent to the Passive Users layer, the outer layer of the structure. Differently from the other layers, the Passive Users layer includes non-technically-savvy users, whose feedback is important for OSS designers.

> "You know, we always get feedback, but feedback from a certain subsection of people. We hear a lot from tinkers, but necessarily from the **average desktop user**. We don't really get feedback from normal people, people that just want to get their jobs done. This is bad, because their feedback is really useful too, especially if we want to make software for a wider audience. People in our community can be really loud about certain problems, but they also have a strong technical bias. Is that something most of our user base run into? There's no way to know. A big challenge is that we currently hear mostly from one side." (P2)

> "There are some people that actually have friends and family members that use our software, and I often get feedback from them. Yesterday, one friend came to me and said: "So, my wife is using [project name] and she wasn't able to do something she wanted to do, so I'm here to ask you how to do it". I believe this is a *very* important piece of feedback. First of all, because that's a bad experience in which someone was not able to do something. And second of all, because it is a piece of feedback I would not get if I did not have that indirect channel. It's **not an accessible user, but one we definitely reach to**. Making things easy for everybody is very important." (P4)

Additionally, all interviewees mentioned **bug tracker tools**, such as Bugzilla[18] and LaunchPad,[19] as a main source of user feedback. As mentioned before, Bug Review is performed in a daily basis and is one of the most time-consuming tasks of OSS designers.

18 http://www.bugzilla.org/

19 https://launchpad.net/

P1 and P4 observed that feedback coming from bug trackers tend to focus on what P1 called "**usability problems**", and not so much on "interaction and experience problems":

> "We tend to get a certain type of feedback from bug reports, people spotting details that should be fixed. Quite often this is, like, people observing some sort of aesthetic issues, or that something should be renamed or is not consistent with whatever stuff... Another quite frequent is 'I can't find something' or 'Something is not discoverable'... You know, stuff like that(...) Details about the interface itself. I'd say we get a lot of bugs on this kind of **usability problem**, but not so much on actual, you know, interaction or user experience problems. I mean, people having issues with their workflows and that kind of stuff. " (P1)

> "A lot of times I get reports from people spotting minor issues, **often issues that the designers and the developer already spotted through usability inspection**. (...) I mean, those reports are of course important, in special when they are not so obvious, not stuff that we could easily spot ourselves. Those in particular, are very nice. (...) It's rare that I get to spot people talking about their experiences at a higher level." (P4)

Those statements are very closely aligned with the results of our analysis of bug reports in the GNOME project's bug tracker. As previously mentioned (section 5.2), we tagged each bug report in our sample using two different sets of tags, one based on Nielsen's heuristics (Nielsen, 1994b), and another based on the user utterances of CEM (de Souza *et al.*, 1999). This procedure was based on the work of Salgado and colleagues (2006), a comparative study of three distinct HCI evaluation methods, including the Heuristic Evaluation and CEM. According to Salgado and colleagues, the problems observed by Heuristic Evaluation are mostly related to interface characteristics, noting the presence or

absence of general design guidelines. We believe those are related to what P1 referred to as “usability problems”. CEM, on the other hand, proved itself to be more suitable to the identification of interaction problems, revealing issues related not only to the execution of n specific procedure, but also to the logic applied to it. We believe those are related to what P1 referred to as “interaction and experience problems”.

In a sample of 547 bug reports, we were able to apply at least one heuristic tag to 459 bug reports (83.9%), and at least one utterance tag to 205 bug reports (37.5%). These results align with the statements of P1 and P4 with regards to the kind of problems reported the most through bug reports. Besides that, It's worth noting that some bug reports had more than one heuristic/utterance tagged to them, and some had none.

Table 9: Distribution of bugs tagged with Nielsen's heuristics.

Heuristic	#	% from 547
Flexibility and efficiency of use	183	33.5%
Aesthetic and minimalist design	150	27.4%
Consistency and standards	123	22.5%
Error prevention	60	11%
Recognition rather than recall	51	9.3%
Visibility of system status	46	8.4%
Help users recognize, diagnose, and recover from errors	34	6.2%
Match between system and the real world	32	5.9%
Help and documentation	18	3.3%
User control and freedom	15	2.7%

	712

Table 10: Distribution of bugs tagged with the utterances of CEM.

Utterance	#	% from 547
Where is it?	105	19.2%
I can't do it this way.	69	12.6%
What happened?	55	10.1%
What's this?	55	10.1%
Oops!	28	5.1%
What now?	21	3.8%
Why doesn't it?	17	3.1%
Where am I?	15	2.7%
Thanks, but no, thanks	5	0.9%
Help!	0	0
I can do otherwise.	0	0
I give up.	0	0
Looks fine to me.	0	0
	370	

Table 8 and Table 9 present the results of this tagging process using Nielsen's heuristics and the utterances of CEM, respectively. From both tables, it is possible to observe that the occurrences of Nielsen's heuristics are much more abundant than the occurrences of the utterances of CEM. Among Nielsen's heuristics, we observe an outstanding number of occurrences of problems tagged with the *Consistency and standards*, *Flexibility and efficiency of use*, and *Aesthetic and minimalist design* heuristics (Table 8).

Additionally, among the utterances of CEM that were most often found in our analysis, we have *What happened?*, *Where is it?*, and *What's this? (*Table 9). According to Silveira and colleagues (2004) (section 7.1), these three utterances reflect problems associated to operational-level affordances, meaning they are related to "the immediate and individual actions that users need to perform". This definition is somewhat close to what P1 described as "usability problems".

Higher affordance levels include problems associated to "conceptualizations and decisions involved in problem-solving processes" (strategic level) and to "a plan, or a sequence of actions, for executing a certain task" (tactical level) (Silveira *et al*, 2004). It is possible to observe in Table 9 that problems at higher affordance levels were not frequently reported. The exception is the *I can't do it this way* utterance (a strategic or tactic utterance), used frequently in the bug reports in our sample as an *I can't do it this way, but I'd like to* utterance. All bug reports tagged with the *I can't do it this way* utterance described the reporter's wish to be able to accomplish a certain task in a way he is not currently able to.

Summarizing the findings in this theme, OSS designers use multiple sources to obtain feedback: bug tracker tools, social media websites, blog posts, and in-person conversations. However, most of these sources present limitations OSS designers have to deal with, such as being too time-consuming to process, consisting of incomplete pieces of feedback, or being rather inaccessible. Additionally, we observed that bug tracker tools, OSS designer's main source of

feedback, often provide a limited variety of feedback, since bug reports tend to focus on operational issues, and not so much on more abstract and/or broader aspects of interaction.

B.3) Designers' attitude

This theme exposes different attitudes OSS designers adopt when getting feedback: seeker, when actively searching for feedback, and receiver, when obtaining feedback in an unsolicited way.

All interviewees demonstrated to assume a **seeker** attitude when dealing with feedback in the form of **bug reports**. Browsing bug trackers' databases is one of the main activities performed by the interviewees, and something they expend a considerable amount of their contribution time on. Three of the four OSS designers interviewed mentioned being **subscribed to bug feeds**, meaning they are directly notified about bugs submitted against a certain product as P4 explained:

> "I do **subscribe** to our main [product name] bug feed, so every time a new bug is filed, I get a notification on that. So I get a lot of feedback from that already. I review those everyday." (P4)

P4 and P2 also mentioned that they **explicitly ask to be forwarded any bug reports on HCI matters** that they might have missed when browsing through the bug tracker's database. It was mentioned by both participants that this kind of bug report is usually written in **fairly technical language**, so that it becomes

difficult to spot that the bug actually reports an HCI problem:

> "I also **specifically ask developers** to cc me to every bug I might have missed and where they feel a design discussion is needed. That happens a lot when when the reporter is a **fairly technical user**. Some reports are fairly technical, and I don't even know what they are talking about, so those just pass me by, even when they are actually related to a design matter." (P4)

P1, P2 and P4 observed that bug reports get such attention mostly because of its medium, since the bug tracker tools in which bugs are reported allow for easy **engagement of developers**:

> "A good thing about bug reports is that they allow you to **engage the needed parts fairly easily**. I mean, developers already live in [bug tracker tool], and use it to solve a number of issues and coordinate development. The degree of indirection is much smaller when you just solve stuff through the bug tracker." (P1)

Indeed, in our analysis of bug reports, we observed that **48%** of the bug reports in our sample had comments from **developers and designers discussing solutions for the reported problem**. We also noted that developers of the Empathy chat client, for example, used bug reports to foster discussion on the design or redesign of certain features. In these cases, the developer filing the bug report describes the feature in question and explicitly solicits input from a designers, for example:

> "As discussed on IRC, one bug for all my niggles about the update accounts UI. I've attached 2 screenshots, one for editting an online account, one for editting an offline account.

> Some of these probably need an actual designer looking at it :)"

After that, the developer in question lists a number of changes in the user interface that should discussed together with a designer. Another example describes a review of part of the user interface, given the removal of a certain user interface element:

> "Now that we are about to remove the roster's menubar we should consider doing the same in empathy-chat."

> Again, the developer filing the report in question lists a number of user interface changes that should be made, and then asks for a designer's opinion on them. In both bug reports mentioned, designers and developers negotiate, in the bug report page, how the user interface should be implemented.

P2 was the only interviewee to mention other strategies when seeking for feedback. Whenever P2 iterates on a design, he **blogs about it** in order to get feedback before the design is actually implemented:

> "When I have a certain piece of design done, **I'll do a blog post about it**, I'll show sketches that we have for it and ask people "what do you think about this?". So, I get a whole pile of blog comments and private emails from that blog posts and buzz from social media, and I'll go through it. (...) Then I try to do a round of follow-up posts, like, "so this is what you said, this is what I'm trying"." (P2)

When **not referring to bug tracker** tools, the interviewees adopt a **receiver** attitude towards feedback, especially since bug reports consume so much of their contribution time. All the OSS designers interviewed mentioned **not suffering from a lack of feedback**, constantly receiving references, from

other members of the community, to pieces of feedback:

> "Usually, if someone sees a certain piece of feedback on the internet, they will bring it to my attention. It's almost like a network of people looking into this, forwarding feedback to us. This might tends to be a **bit overwhelming** from time to time, since we're not a bunch of designers and people expect us to address at least a big part of it [the feedback]." (P3)

> "I definitely **don't think we suffer from a lack of feedback** at all (laughs). I don't usually need to seek for it, I already get a lot of feedback coming for me from developers and colleagues. Generally speaking, I don't have to ask for it, I just get it all the time." (P4)

It was possible to observe the OSS designers interviewed adopt a seeker attitude towards feedback reported via bug tracker tools, while they tend to adopt a receiver attitude towards other sources of feedback. The ability to easily engage developers in the discussion of a report influences the attitude adopted by the interviewees towards feedback.

C) Using feedback

This theme explores aspects of how the OSS designers interviewed interact with and manage feedback obtained from different sources.

C.1) Motivations

This theme uncovers what motivates the interviewees to work on the problems exposed by a certain piece of feedback. The interviewees were unanimous in stating that, in general, the **quality of a report does not affect their motivation**, but rather their ability to work on a given issue. An exception to

this is feedback where the reporter adopts an **aggressive or negative** attitude towards the designer or the community. This kind of feedback was mentioned to be quite **demotivating** for the OSS designers interviewed, and yet quite frequently observed:

> "When feedback is just openly aggressive, which it is a lot of times, I find it really demotivating to work on. Stuff that just says, like, "this version sucks, I can't use this". **The attitude is really demotivating and the content itself is not even helpful** so that I can do something about it. When people just say "why did you made this thing like this? This is really stupid, I hate you", you know? It is just plain offensive. (P1)

> "One thing [that is demotivating] is the **emotional outbursts that you see online**. (...) More serious than that [in relation to another kind of feedback] are the expressions of anger, upset or disappointment, or the personal accusations about the designers or community, that we frequently encounter around. These make it difficult to engage in a conversation about what the actual issues are. Personally speaking, I would also say that they can make designing in the open an **emotionally draining experience** and far less attractive than it should be." (P4)

Apart from that, the OSS designers interviewed were also unanimous in stating that the **amount of users affected** by an issue is another strong motivational factor in terms of prioritizing issues. One reason for this is that **wider adoption** of OSS is of great concern to the interviewees, especially because of their commitment to the Open Source ideology and to making software available and usable to everyone.

> "I mean, if a **lot of people** are having a problem, that's motivational. But also... Like, if people are dropping out, stopping to use the software because of this

issue, that's also a *big* motivator." (P1)

"It [my motivation] really depends on **how many people** it [the problem] affects. You know, how many people we perceive it will affect, and how badly it affects them. Those are very important parts of what problems to solve first, because we want to **make our software, free software, usable for as many people as possible**." (P3)

"If it's something that may **affect a lot of people**, than I add it to my personal list of tasks, and might mention it to other designers and developers." (P4)

At the same time, both P1 and P4 mentioned that feedback that reports **problems they relate to** tends to catch their attention:

"[asked about motivations to work on a given issue] Well, I'd say it grabs my attention when it's about **something I personally identify with**." (P1)

"Generally speaking, when I review them [bug reports], what I'll do is check on the ones that look interesting to me, meaning something that affects the user experience in a way **I can relate to**, something I can connect to." (P4)

This interest in reports that describe issues with which they personally identify themselves might be associated with Raymond's (1998) archetype of developers "scratching their personal itches" in OSS projects.

Another strong motivational factor for the OSS designers interviewed is how the reported problem fits into the overall "**design vision" and the development priorities** of their respective projects:

> "The way we have been prioritizing problems... I mean, this is more of a **release decision**. We really need to focus on certain aspects or parts of the software from release to release. So, for example, when I go through a list of usability bugs, I sort them, I make sure all the release-relevant bugs are at the top of the list and that we get to work on them first." (P2)

> "Well, you cannot solve every bug, you have to work on certain ones that fit into **your priorities and your scope or your design basis, your design vision** (...) Because of the nature of Open Source, you often get bug reports on really specific uses and expectations. There might be bugs that are possibly genuine, but then they are not within our power, or willingness, to fix, considering the overall approach that we're taking." (P4)

Finally, as mentioned before, P3 stated that problems originated from **partial implementation** of their designs also tend to catch his attention, because they represent a way for him to push developers to complete the development of a certain feature or software.

In conclusion, when choosing what issues to work on, OSS designers take into consideration the amount of people an issue affects and how solving this issue fits into the project's priorities and design message. Additionally, feedback reported in an aggressive/negative tone was observed to be demotivating to the interviewees and discouraging when it comes to designing in an open and transparent manner.

C.2) Obstacles

This theme exposes the obstacles the OSS designers interviewed face, when managing and addressing problems reported through feedback. The

interviewees were unanimous in observing three main obstacles in dealing with feedback. The first one is related to the **negative and/or aggressive attitude** adopted by reporters frequently, previously mentioned in theme . The interviewees mentioned that those reports tend to be very **unspecific about what the actual issues are**, offering little useful information for them. Engaging with the reporter to gather more information about the problem when the reporter adopts that kind of attitude seems to be ineffective, according to the OSS designers interviewed.

The second major obstacle faced by the interviewees exposes a **mismatch between the information they need in order to understand and work on a given problem, and the information that is actually provided by reporters**. It was observed by all interviewees that, more often than not, reports are unclear about what the actual issue is, missing important information for its characterization.

> "[when asked about obstacles for finding useful feedback] Bugs that are generally unhelpful, but well-intentioned, are bugs that **don't have enough information** on what the person was trying to do, or even what exactly happened. Like, we had a bug that I was going through this week, and the guy was saying "oh, it crashed when I did this", but it didn't have enough information for me to understand. I mean, crash? How did it crash? There's a million ways a thing can crash." (P2).

> "One of the things that make my life hard is what I describe as "**incomplete reports**". So, it's people not clearly explaining what are the things they were trying to achieve, what the exact problems that they faced are, etc. (...) That makes it really hard for me to act on those problems." (P4)

In fact, in our analysis of bug reports, we noted that interaction with the reporter, for the purposes of clarifying the reported problem, was attempted in **41.8%** of the bug reports in our sample. It is worth observing that, when this analysis was conducted, 19.5% of the bug reports in our sample had no comments. It is not possible to determine the reasons for the lack of comments in those reports, but, since they are not marked as closed, interaction with the reporter might still occur.

This mismatch might also be noted in reports that consist of a **solution to a problem**, missing information and details about what the problem is and how it was observed. These reports represent a subset of the "incomplete" reports, since they also fail to provide information relevant to the problem's definition, as P2 and P3 explains:

> "Another type of bug that we get *a lot*, that is well-intentioned, but not useful at all, is like, say... "I want you guys to do it this way" or "**I think X should be Y**", and not explain why or what they were trying to do or what the problem was that they are trying to solve with this proposal." (P2)

> "A lot of times, people that file UX bugs don't necessarily know that much about UX, but they think they do. They tend to **give me a solution, instead of a problem**. Like, "this should be X, instead of Y". And I'm like... "Well, maybe". Usually I have to try and reach out to that person and, hm, like "why do you think it should be like this?". You gotta understand why they think this is better or important and, especially, what is the underlying problem that they think they are solving." (P3)

This tendency of reporters to provide a solution instead of a problem was

confirmed by our analysis of bug reports. For each of the bug reports in our sample, we noted either the reporter described a problem, a solution or both. Our results suggest that 19.3% of the bug reports analyzed described a problem, 39.5% described both a problem and a solution, and **41.2% described only a solution**.

The third obstacle the OSS designers interviewed mentioned is related to **understanding how different reporters express their issues**, as P4 explains:

> "It's *very* hard to identify that two usability bugs are actually the same, because they are very different from one to another. I mean, [bug tracker tool] actually offers facilities to identify if two bugs are similar, but with usability bugs, that's virtually impossible. **It's all on how someone express themselves and this varies a lot from person to person**. People use very different terms for the same things." (P2)

> "Usually, it is **pretty hard to understand what people are describing**. It's different from when you're talking to other designers and developers, and we have this kind of common language, and we describe things mostly with the same terms. (...) Often, I have to go into fairly lengthy conversations with reporters just to understand what are they talking about." (P4)

We observed evidence of this obstacle in bugs considered duplicates of each other in the GNOME project's bug tracker. During our analysis of bug reports, we observed occurrences of bugs with very distinct summaries and descriptions, which were identified reporting the same issue. For example, the following set of bugs, represented by the bugs' summaries, were marked as duplicates of each other:

- Don't use “ctrl” modifier for delete action
- Change the Ctrl+del key back to Del and use notifications instead
- Add dconf setting to customize “Move to Trash” key binding
- nautilus uses CTRL+Delete instead of Delete
- Ctrl-Delete and Shift-Delete make it very easy to accidentally delete a file instead of moving it to the trash
- delete key does not work
- Can't delete with “del” key after setting file deletion shortcut to “del”
- Pressing delete while renaming file moves it to Trash.
- gtk_window_activate_key messes up key bindings.

It is possible to observe that the aforementioned bugs are **phrased in very distinct ways**, even though, according to developers/designers, they all **report the same issue**, the fact that it is not possible to delete files by pressing just the “Delete” key.

This obstacle is aligned with the findings from Furnas and colleagues' study (1987), focusing on the phenomenon of word choice for UI objects by end users. Observing spontaneous choice for describing elements of five different application-related domains, they observed very high variability rates on the terms chosen by users to describe UI elements and interaction actions.

Additionally, P3 also observed that, given the lack of opportunity to apply observation methods, it might be hard to identify **when their design**

solution is actually succeeding in its goals:

> "I don't have resources to do good user testing. So I really rely on people to bring the problems themselves. I can't get proactive, I just have to sit and hope people care enough to actually give feedback. And **generally it's hard to know when your design is succeeding**, but it's easy to know if it's failing. Because people will complain more than they will praise. So, it's almost like "no news is good news". If you don't hear anything back from people, hm, so, "this is probably fine, because no one is complaining about it". But, you know, sometimes it just doesn't bother people enough for them to complain about." (P3)

In general, the major obstacle the interviewees face when dealing with feedback is related to this mismatch between the information they need to have in order to understand and address an issue, and the information that is actually provided by users. This highlights the need for supporting users in creating reports that are more closely aligned with OSS designers' needs.

6.2 RQ2: What are the kinds of information OSS designers need in reports of HCI issues?

As previously mentioned, we analyzed the transcripts of the interviews twice, each time with a different research question in mind. For the second analysis, we focused our efforts in exploring the kinds of information that OSS designers look for in user feedback. As explained in section 5.1, we used Thematic Analysis for creating themes that described the kinds of information the interviewees considered necessary in order to perform their tasks as OSS designers. We phrased these themes as questions from the OSS designers to the

user reporting a problem, as follows:

- what were you trying to do?
- why did you want to do it?
- what did you do?
- what happened?
- what were your expectations?
- what are you running?

In the next paragraphs, we describe each of those themes, illustrating them with snippets from the interviews with OSS designers.

what were you trying to do?

This theme reflects the need for information regarding the **broader goal of the user** when he started the interaction with the system. All the interviewees mentioned this as a critical piece of information to understand how their design solution is failing users.

> "Some of them [reports] are like "I don't like this version because it doesn't have enough settings". I want to ask them "What is it that you are actually trying to **achieve**?". I personally don't find "number of settings" to be a good UI metric. Tell me, what is it that you need to do?" (P1)

> "Users, when they file usability bugs, tend to talk about what they want the UI to be, rather than what they are actually trying to do. (...) Frequently, we try to get in touch with the original reporter to get more information about **what is that they were trying to do**, because that has to be the drive of the solution. You don't want to solve the problem that is not the problem you have." (P2)

> "One important part of it is figuring out **what the person was trying to do,**

> **what was their goal**. (...) I mean, someone says "Such and such sucks!" and then you need to engage and ask "OK, tell me what were you trying to do", etc, etc. By the end of that, you may have identified a string of issues that needs to be fixed." (P4)

All the interviewees stated "what were you trying to do?" as the most frequently absent information in reports. As mentioned before, more often that not, reporters focus on providing solutions to the problems they experienced, omitting the rationale behind their "design decision". P2 and P4 explain that those design suggestions are often not helpful, since reporters are frequently not informed of technology restrictions and of the broader design vision.

> "It's great that everyone wants to be a designer, but they don't necessarily understand the technology underneath and how it works." (P2)

> "Some things are just hard to solve by the nature of the design. And also by the nature of the engineering itself, I guess. Usually, reporters are clueless about those things." (P4)

why did you want to do it?

This theme is related to information on the user's personal reasons to be trying to achieve a certain goal, revealing data on the **context** that triggered a certain interaction with the system. Information under this theme addresses users' motivations and constraints, and is often not related to tangible aspects of the system's user interface.

> "There's a number of reasons why someone cannot achieve a certain goal, and it's hard to understand it without **the context, the motivations** for the user to be doing a certain thing." (P1)

> "The most helpful ones are the ones that actually say, like, "I wanted to do this because of blah". (...) I mean, it's good to know **why someone would need to do something**, because you can better support them." (P2)

> "We get a lot of reports that are like "I want it to do this", with no rationale of **why would you want to do that**." (P3)

> "We got a bug report saying "There's no way to view contacts smaller, and that's something that should be included" and that was not particularly helpful, because I did not knew why that person wanted to have their contacts smaller. And then I asked and he said "Well, it's because I have a lot of contacts, many of them do not have avatars and my monitor is small. So I need them to be smaller to see who's online". That **context information** was really valuable to me". (P4)

Some of the interviewees mentioned that this kind of information enables them to come up with design solutions that not only solve users' problems, but also provide a pleasurable experience to them.

what did you do?

Data under "what did you do" often takes the form of a **step-by-step description** of what the user did during the actual interaction with the system. Most OSS designers interviewed mentioned they would like reporters to "**tell a story**" about their interactions with the systems. Some of the interviewees added it would be helpful if this story included not only the actual operations performed by the reporter, but also the reasons why the reporter thought those were the correct actions to be done.

> "[talking about a helpful report] And another thing is that he gave me **data about what he did**." (P1)

> "**It's great when reports are very narrative**, if that makes sense. "Hey, I was trying to use the installer, and I wanted to set the time locally and bla bla, so I did this and I couldn't figure out...". You know, it is very much like a story." (P2)

> "The **full story is very useful**. Like, this is what I wanted to do, this is where I started, this is what I did, this is where it all went wrong, this is how I tried to work around it... I need to understand the problem, the steps the person made and all." (P3)

It was also mentioned that this step-by-step description of the reporter's interaction with the software should also include the reporter's attempts to work around the experienced issue. P1 and P2 mentioned that this kind of information is helpful for understanding the reporter's "**mental model**", which they mentioned as a way to get to know their user base.

> "(...) [talking about a helpful report] he describes the different paths he took to try to solve the problem (...) He explained every single action he took along the way, and this tells me what is his **mental model**, if that makes sense. Like, where he goes in order to work around certain issues. (...) It's interesting to understand what he's thinking along the way." (P1)

> "When users tell you what they tried to do in order to solve an issue, you have a broader idea of where you're succeeding and where you're failing. **It's often very good if you can understand where the user is coming from in the design**, if you get to know where he has been to." (P2)

The interviewees' opinions conflicted, however, when it came down to how the whole "story" should be partitioned when taking the form of reports. Half of the interviewees thought a single report about the whole experience

might give them a better chance to fix the design at a broader level.

> "Problems like "this label is broken" are easier to fix, and those are the things we usually have in Bugzilla, but perhaps I'm missing the chance to fix the whole workflow, because the user just reported part of his experience. Then I'll fix this stuff and all, but perhaps I'm just making the experience less unpleasant, but it is still broken." (P1)

The other half, however, stated to prefer reports to be focused on smaller problems, claiming them to be easier to manage, especially in terms of interaction with developers. Designers in accordance with this point-of-view, however, seemed to be influenced by limitations and characteristics of their current bug tracker tools of use.

> "One thing about bug trackers is that having lots of separate issues in a single report makes it hard to discuss and hard to manage as well. The threads tend to go to a bunch of different directions and ending up in no conclusion." (P2)

> "What I try to encourage is to people to report issues as they experience them. If someone is experiencing something that's related to a bunch of different issues, I encourage this person to file a set of bugs. But that may be a limitation of our current bug tracking philosophy and tools, I guess." (P4)

P1 mentions that a combination of both reporting styles might also be beneficial, as it enables designers to have a broader view of the experience while creating smaller action items for developers:

> "[taking about a helpful report] It's interesting that his experience had to do with several components that work together. So, it affects the shell, the search, the online accounts setup, the chat itself. The chat client itself is not completely

> broke, it's the broader experience that is, and we have a lot of attack points. So [name of the reporter] tells me this entire story [on Google+], which is highly valuable to me, but also opens a set of bug reports at Bugzilla for each of the smaller problems, which is very helpful for us to coordinate work with developers." (P1)

what happened?

This theme is related to the **actual issue perceived** by the user. The interviewees mentioned that the biggest issue with this kind of information is that reporters tend to be quite inaccurate or unclear when describing interaction breakdowns. It was observed that reporters often describe their experienced issues as "X was broken", without any specification of what it means to say that X is broke and what are the observed symptoms for it.

The interviewees mentioned that the most helpful reports are the ones that expose the experienced problem in detail, describing how it was observed in the interface. They explained that the main challenge they face when trying to gather this kind of information is to **understand what the reporters are describing**.

> "Often you may have fairly lengthy interactions with people just **trying to understand what they are talking about**." (P1)

> "Usually, it's pretty **hard to understand what people are describing**. It's different from when you're talking to other designers and developers, as we have this kind of common language, and we describe things mostly with the same terms." (P2)

> "Sometimes they [reporters] might describe a problem, and I try to reproduce it in my machine, but **it's not very clear what they were experiencing**." (P4)

As a workaround to this problem, some of the interviewees mentioned that screen shots of **what the reporter is seeing** in the moment of the problem are quite helpful.

> "Some **visual description** of what happened is a lot more efficient. To show it to you instead of describing it in words." (P1)

> "[talking about a helpful report] The thing that was the best about his report is that he gave me **screen shots** of what exactly he was talking about." (P2)

> "A fairly good thing to know is what's actually happening, so, **what someone actually seeing in their screen**. (...) Recently, I was checking one of those social media sites and someone said "Oh, I don't like this design, it makes me scroll too much". Well, as far as I was aware, that design was meant to reduce scrolling in that area, not increase it. The question then was "What are they actually seeing? Why are they scrolling so much?"" (P4)

In the case where the reporter is describing an unpleasant experience with continuous use of the software, interviewees mentioned they quite often are faced with unspecific reports that mostly announce that something is hard or difficult or frustrating, without any further explanation for what motivated such statement. P3 observed that, in those cases, it is helpful for them to understand what **patterns in the software** trigger the reporter's negative emotions.

> "It's very unhelpful when people do not state what the problems are, what are the sources of their frustrations. Some say they are having a bad experience with the software, like, "This doesn't work for me". Why not? It's important for me to **understand this person's workflow and what are my patterns in failing**

this workflow. Recurring things, you know?" (P3)

what were your expectations?

This theme describes information related to the reporter's expectations in terms of the system behavior. The interviewees mentioned that it is often helpful when the reporters describe **what they expected to happen** when the experienced problem occurred, highlighting how the system frustrated these expectations.

> "What did you expect to happen? You know, what should have happened? **The expectation is usually very revealing**. (...) This is where I think the best information comes from, from the expectations" (P3)

> "So, I lot of the feedback I get is "Oh, I think X should be like Y", and we don't know if that's because that reporter just thinks that's a better idea, or **if it's something related to their actual expectations, their uses and experience**. So, the difference between "I think X should be like Y" and "I tried to do X, and I expected Y to happen, but then Z happened and I wasn't able to accomplish X"... That definitely would be more useful." (P4)

> It was observed that this kind of information might also be found when the reporter describes what triggered the interaction problem, so that the designer have information about an interaction strategy his design failed to accommodate or communicate. In a similar way, when the reporter describes the ways in which he tried to work around an issue, the designer might also have information on recovery or secondary interaction strategies.

Some of the interviewees mentioned that, by understanding a reporter's expectations, most of the time, they also get to understand the **reporter's "mindset"** and experience.

> "I wouldn't say figuring out who the users are, the profile of the users matters so much. It's more the different kinds of goals, habits and mindsets people have

> when trying to do things. **You usually get a sense of those things by knowing the users' expectations.**" (P4)

what are you running?

The interviewees stated that it is important for them to know with which **version of the software** the reporter experienced a certain issue. This theme started as a general descriptor for technical information, becoming exclusively about software version, with the refinement of the set of themes. The interviewed designers mentioned that often this is the only technical information needed in order to grasp a described HCI problem. They explained that this piece of information is relevant given the short release cycles used in OSS development, and the fact that software may considerably change from one version to the other.

> "Incomplete [reports] also means people not telling which version of the software they are using. Often we get reports saying "This version sucks", and then, when we figure out which version is this, we just tell the guy "Well, this is fixed in the new version"**. The version also helps me understand what the user is talking about**, you know? Frequently, things are added or removed from one version to another. If the user talks about an element in a version, and I think he's talking about another version, then we might end up not understanding each other at all." (P1)

> "Generally, I would say **knowing the software version is one of the most important things**, especially because we're always at this stage of continuous evolution of the software, so there are quite a lot of changes and refinements to the experience from version to version." (P4)

The set of technical information needed from reporters might vary depending on the **domain of functionality of the software**. P1 and P2

illustrate this situation:

> "**Because of the nature of our project**, knowing the distributions is also quite critical, since, you know, some distros also make some critical modifications to our thing." (P1)

> "[talking about a helpful report] He told me about his machine , like "I'm doing this in a VM, it has 2CPUs, it has this much RAM...". **In the case of the [specific piece of software], this would have been the first thing I would have asked** him after I reading his post." (P2)

6.3 Summary

In this chapter, we reported several findings related to the role of reports of HCI issues in OSS designers' activities, noting that they are an important input for them to iterate on their designs. We also reported many obstacles OSS designers face when dealing with reports of HCI issues, among which we find a mismatch between the information provided by reporters and the information OSS designers need in order to act upon reported problems. In order to address this mismatch, in Section 6.2, we elicited the kinds of information OSS designers need to understand and solve HCI issues.

In chapter 7, based on the findings of this chapter, we present the design and evaluation of a form for reporting HCI issues. This form was designed to address the aforementioned mismatch, based on the information needs identified.

7 Improving Reports

In order to address RQ3, “how can we support end users in creating reports of HCI issues that meet OSS designers' information needs?”, we propose a form designed to support users in crafting reports of HCI issue that match the needs of OSS designers identified in section 6.2. This chapter presents the design rationale behind this form, and a study conducted to evaluate its ability to elicit the necessary information from users, through the report of HCI issues.

7.1 Design

The form was designed according to a Semiotic Engineering approach, and based on the work of Silveira and colleagues (2004), which presents a method for building online help systems based on design models.

As explained in section 4.2, Semiotic Engineering views the user interface as a message sent from designers to users, representing the designers' solution to what they believe is the users' problems. According to this theory, it is essential that users understand the designer's message so that they may better use and take advantage of an application. Silveira and colleagues (2004) advocate that users should be able to more precisely express their doubts about the designer's

message and needs. Their work aims to enable designers to “anticipate such doubts and needs, and to organize their response accordingly” through the design of help systems.

We are inspired by Silveira's idea that users should be able to express their doubts and needs. However, our approach does not aim to anticipate the designer's response to them. Instead, we aim to enable users to express such doubts and needs, so that the designer can then review his message having them in mind. By observing the findings from our interviews with OSS designers (section 6.1) from a Semiotic Engineering perspective, it is possible to note that this iterative review of the designers' message based on feedback from users already takes place within OSS projects. Our goal is to support this process by providing an HCI issue report format aligned with the information OSS designers need, according to the findings in section 6.2, in order to review their message. In comparison to Silveira and colleagues' work, we aim to enable users to anticipate the designers' needs, instead of enabling designers to anticipate the users' needs. We are motivated by the fact that a lot of the feedback OSS designers receive is incomplete in terms of what they need to know to address HCI issues, and that this makes the aforementioned iterative process inefficient, as observed in section 6.1.

Our initial approach to this HCI issue report format was based only on the types of information needed by OSS designers, as identified through our interviews with OSS designers (section 6.2). We translated the six types of

information identified into a form consisting of six questions, followed by examples of how to respond them (Table 10). The examples are a crucial part of the form design, since they illustrate what is expected from each of the questions, which the questions only might not be able to communicate. This way, they are not an additional resource to the form filling, but an important part of its composition.

Table 11: Mapping between types of information (section 6.2), and initial form questions.

type of information	**question** ***example***
what were you trying to do?	what were you trying to achieve? *I was trying to copy some videos to my pen drive.*
why do you want to do it?	why were you trying to achieve that? *So I could watch them on my friend's computer, since I cannot plug my computer to the TV, as it doesn't have an HDMI plug.*
what did you do?	could you tell me, step by step, how were you trying to do it? *I opened my Downloads folder, where the videos were, selected them and then pressed Ctrl+C to copy. Then, I opened my pen drive's directory and pressed Ctrl+V to paste the videos there. I waited until the transfer was over and ejected my pen drive, by pressing the eject icon close to its name in the side bar.*
what happened?	what went wrong? *I got a notification saying "Writing files to pen drive" or something like this, but was never told when it finished.*
what were your expectations?	what did you expect to happen? *I expected a clear sign that I could remove my pen drive without damaging my files, like a notification or something.*
what are you running?	what version of the software are you using? *GNOME 3.6 and the Nautilus file manager (3.6 too).*

An inspection of this initial version of the form revealed its bias for eliciting information about problems associated to the system response, seeming unsuitable to problems associated to a user's attempt to express his communicative intent. Because of this, our final approach draws on the user utterances of the Communicability Evaluation Method (CEM, presented in section 4.2.1) to propose an HCI issue report format, having in mind the different kinds of breakdowns that a user might face when interacting with a system. We also take into consideration the additional user utterances proposed by Silveira in colleagues (2004) (Table 11) in order to address users' procedural and motivational doubts, which are not addressed by the original set of utterances of CEM. Table 12 presents the whole set of utterances considered for designing our HCI issue report format.

Some of the existing CEM utterances were considered inadequate for the purposes of a form for reporting HCI issues. For example, a user would never utter *Looks fine to me.*, since this utterance is associated with the user's inability to recognize a problem with the expected results. Another unused CEM utterance is *Help!*, since it does not describe an actual issue, just a scenario where the user explicitly solicits information through the use of a help system.

Table 12: Help-specific utterances proposed by Silveira and colleagues (2004).

user utterance	**illustrative symptoms**
Where was I?	The user needs to retrace his steps in order to understand the state in which he currently is.
Why should I do this? What is this for?	The user doesn't understand the reasons underlying certain instructions or the utility of a certain task.
Who is affected by this? On whom does this depend? Who can do this?	The user needs information about the work processes and roles of the application.
How do I do this?	The user doesn't know how to perform a certain task in an application.
Is there another way to do this?	Comprises both the *I can do otherwise* and the *Thanks, but no, thanks* utterances of CEM.

Table 13: Final set of utterances considered for the design of our HCI issue report format.

CEM utterances		
Where is it?	What now?	What's this?
Oops!	I can't do it this way.	Where am I?
What happened?	Why doesn't it?	I give up.
I can do otherwise.	Thanks, but no, thanks.	
Help utterances		
Why should I do it?	What is this for?	Is there another way to do it?
On whom does this affects?	On whom does this depend?	Who can do this?
How can I do this?	Where was I?	

Our strategy for designing our HCI issue report format consisted of adjusting the six questions, together with the example scenario, to each of the utterances we selected. During this stage of the design of the format, we observed that, indeed, it was not possible to describe the problems characterized by all utterances using the same set of questions. However, we also observed that different groups of utterances worked well with different sets of questions. This led us to group the utterances by the affordance level (operational, tactical or strategic) in which they occur, following the analysis of Silveira and colleagues (2004), presented in Table 13. We argue that problems at different affordance levels require different tools to be described, since in Semiotic Engineering a designer has accomplished a successful communication with the users when they can perceive the intended application affordances (Silveira *et al.*, 2004).

The final version of the form uses three sets of questions, whose main difference to each other revolved around the question associated to the "what happened?" and "what were your expectations?" information types. This piece of information is critical to the form because it represents the reporter's opportunity to more precisely express misconceptions on the designer's message. The decision on which set of questions to use depends on which of four labels the reporter chooses to characterize his problem. The labels are: *I don't like the way something works*, *I can't figure out how to do something*, and *Something is confusing or unclear* (plus the label *Other*).

Table 14: User utterances per affordance level, according to Silveira and colleagues (2004).

affordance level	**user utterances**
operational	Where is it? Oops! Where am I? Whom does it affect? On whom does this depend? Who can do this? Where was I?
operational, tactical	What's this? What happened? What now?
tactical	How do I do this? I give up.
tactical, strategic	Why doesn't it? I can't do it this way.
strategic	Why should I do this? What is it for? Is there any other way to do this? I can do otherwise. Thanks, but no, thanks.

The labels were chosen as a way to represent the utterance groups, after their aggregation by affordance level. The final grouping of the utterance is very similar to Silveira and colleagues' (2004) analysis presented in Table 13. The user utterances groups and their differences to Silveira and colleagues' work is presented as follows:

I don't like the way something works.	
Why should I do this? (strategic)	*What is it for? (strategic)*
Is there another way to do this? (strategic)	*I can do otherwise. (strategic)*
Thanks, but no, thanks. (strategic)	*I can't do it this way. (strategic, tactical)*
Oops! (operational)	

This utterance group is related to problems associated to strategic-level affordances, which means they are related to "conceptualizations and decisions involved in certain problem-solving processes and in the embedded technology" (Silveira *et al*., 2004). These utterances are particularly important from the perspective of Semiotic Engineering because they explicitly express the reporter's disagreement to what the designer's message states. For example, a report related to the *I can do otherwise.* or *Thanks, but no, thanks.* utterances may express that a certain interaction path feels sub-optimal. The *Oops!* utterance, even though it is not classified as a strategic-level utterance, was added to this group because it represents an instant mistake, that the user rapidly recognizes and try to repair. The reporter might then manifest that he doesn't appreciate how the system's interface led him to do that mistake.

The form for reporting problems under *I don't like the way something works* is described as follows*:*

1. What were you trying to achieve?
2. Why were you trying to achieve that?
3. Could you tell me, step by step, how were you trying to do it?
4. What was the problem?

5. How did you expect to do it?
6. What version of the software are you using?

I can't figure out how to do something.	
Why doesn't it? (strategic, tactical)	*How do I do this? (tactical)*
I give up. (tactical)	*What now? (tactical)*
Where is it? (operational)	

This utterance group is related to problems associated to tactical-level affordances, which means they are related to "a plan, or sequence of actions, for executing a certain task." (Silveira *et al*., 2004). This group consists of utterances describing situations where the reporter was not able to express his communicative intent in terms of an interaction plan. The *Where is it?* utterance, even though it is not classified as a tactical-level utterance, was added to this group because it describes a situation where the reporter was unable to find an interface element suitable to the expression of his communicative intent.

The form for reporting problems under *I can't figure out how to do something* is described as follows*:*

1. What were you trying to achieve?
2. Why were you trying to achieve that?
3. What went wrong?
4. Could you tell me, step by step, how were you trying to do it?
5. What version of the software are you using?

For this utterance group, the question related to the information type "what were your expectations" was omitted. We understand that, when the user

was not able to do something through the system, their expectations are expressed through what they attempted to do in order to achieve their goal (the "what did you do?" information type, expressed by the question "Could you tell me, step by step, how you were trying to do it?).

Something is confusing or unclear.	
What's this? (tactical, operational)	*What happened? (tactical, operational)*
Where am I? (operational)	*Where was I? (operational)*
Whom does it affect? (operational)	*On whom does this depend? (operational)*
Who can do this? (operational)	Where is it? (*operational*)

This utterance group is related to problems associated to operational-level affordances, which means they are related to "the immediate and individual actions that users need to perform" (Silveira *et al*., 2004). This group consists of utterances describing situations where the reporter was not able to understand what the system's interface is communicating.

The form for reporting problems under *Something is confusing or unclear* is described as follows*:*

1. What were you trying to achieve?
2. Why were you trying to achieve that?
3. Could you tell me, step by step, how were you trying to do it?
4. What went wrong?
5. What did you expect to happen?
6. What version of the software are you using?

For this utterance group, the question related to the information type "what were your expectations" is phrased differently from the question related to the same type for the *I don't like the way something works* group. For *Something is confusing or unclear*, the question is related to a description of what the reporter expected the system's expression to be. For *I don't like the way something works* group, the reporter is then expected to describe how he expected or would prefer to accomplish or do something.

The complete form, with its example scenarios, can be found in the Appendix C of this dissertation.

7.2 Evaluation

This section describes the study conducted in order to evaluate our form's effectiveness in eliciting the information OSS designers need in reports of HCI issues – according to the types information described in section 6.2.

7.2.1 Procedure

During a period of 16 days, participants in the study were invited to report any HCI issues they experienced with software using our form, implemented in the Polldaddy[20] survey platform. We did not request participants to use any specific software for the study; they were welcome to report HCI issues experienced with any software they used, including their operational system or desktop environment of choice, or any desktop, web or mobile application. We also made no restrictions in relation to the use of

20 http://polldaddy.com/

software registered under open source licenses, since our interest is at the needs of OSS designers, but not at the use of OSS.

Our initial plan was to compare the results of this evaluation to what we observed with our analysis of bug reports from the GNOME bug tracker (Section 6.1). For that purpose, we tried to recruit for this study participants who had filed at least one of the bug reports in our sample, but, unfortunately, only four of the invited reporters accepted to participate. Because of that, we also recruited participants by sending invitations to open source and general purpose mail and Facebook groups. We understand that, with this decision, the samples for this study and for the analysis of bug reports are not comparable. However, while we do not draw any conclusions based on comparing both studies, we still took the opportunity to observe how the form influenced the reporting practices of the four participants who also reported bugs in the GNOME bug tracker. We will refer to them as P1 to P4.

During the 16 days of the study, 26 participants from a variety of backgrounds (Table 14) volunteered a total of 45 reports of what they considered to be HCI issues. These reports were qualitatively analyzed, comparing their contents to what we identified as the information needs of OSS designers. We also tagged the reports using two different sets of tags, one based on Nielsen's heuristics and one based on the utterances used to design the form, similarly to how we tagged our sample of bug reports from the GNOME projects' bug tracker (Section 5.2).

Table 15: Profile of the participants in the study ordered by number of reports .

# reports	occupation	Educational background
8	Researcher	Biology (Doctoral Degree)
4	HCI Designer	Arts and Graphic Design (Bachelor's degree)
3	Linux Technical Engineer	Computer Science (Master's degree)
3	Researcher	Physics (Doctoral degree)
2	Administrative Assistant	High School
2	HCI Designer	Computer Science (Master's degree)
2	Managing Director	High School
2	Student	Law (Bachelor's degree)
2	Student	Political Science (Master's degree)
1	Economist	Economics (Bachelor's degree)
1	Geophysicist	Physics (Bachelor's degree)
1	HCI Designer	Computer Science (Doctoral degree)
1	Legal Technician	Human Resources (Master's degree)
1	Producer	Journalism (Bachelor's degree)
1	Professor	Business Administration (Doctoral degree)
1	Professor	Business Administration (Master's degree)
1	Student	Chemical Engineering (Bachelor's degree)
1	Student	Chemical Engineering (Bachelor's degree)
1	Student	Civil Engineering (Bachelor's degree)
1	Student	Computer Science (Bachelor's degree)
1	Student	Computer Science (Bachelor's degree)
1	Student	Law (Bachelor's degree)
1	Student	Literature (Bachelor's degree)
1	Student	Mechanical Engineering (Bachelor's degree)
1	Student	Political Science (Bachelor's degree)
1	Student	Political Science (Bachelor's degree)
1	Student	Political Science (Bachelor's degree)

Apart from that, we administered the National Aeronautics and Space Administration Task Load Index (NASA-TLX) (Hart and Staveland, 1988) procedure to assess the workload of the task of filling our form. The NASA-TLX uses six dimensions to assess the workload of a given task: M*ental demand*, *Physical demand*, *Temporal demand*, *Performance*, *Effort* and *Frustration* (Table 15). Following the procedure, after the 16-day period during which the participants used the form, we requested them to fill a second form consisting of six bipolar scales, each corresponding to one of the NASA-TLX dimensions, divided from 0 to 100 in increments of 5. This form also consisted of 15 paired comparisons between the six dimensions. Paired comparisons require the participant to choose which dimension was perceived as more relevant to the workload of the task. The number of times a dimension is chosen as more relevant is the weight of that dimension for the task for that participant. The procedure uses these weights to combine the scale ratings into a global score representing the overall workload for that task according to a participant.

In the form we used for administering the NASA-TLX procedure, participants were also asked if they had any comments on the form used for reporting HCI issues.

Table 16: Workload dimensions in the NASA-TLX procedure (Hart and Staveland, 1988)

dimension	**endpoints**	**description**
Mental Demand	Low – High	How much mental and perceptual activity was required (e.g. Thinking, deciding, calculating, looking, search, etc.)? Was the task easy or demanding, simple or complex, exacting or forgiving?
Physical Demand	Low – High	How much physical activity was required (e.g. pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?
Temporal Demand	Low – High	How much time pressure did you feel due to the rate or pace at which the task or task elements occurred? Was the pace slow and leisurely or rapid and frantic?
Performance	Good – Poor	How successful do you think you were in accomplishing the goals of the task set by the experimenter? How satisfied were you with your performance in accomplishing these goals?
Effort	Low – High	How hard did you have to work (mentally and physically) to accomplish your level of performance?
Frustration	Low – High	How insecure, discouraged, irritated, stressed, and annoyed versus secure, gratified, content, relaxed, and complacent did you feel during the task?

7.2.2 Results

In this section, we present the results of this evaluation from three different perspectives: accordance to the information needs of OSS designers, nature of the problems reported, and workload imposed on reporters.

Information needs of OSS designers6.2

Table 17: Amount of reports that do not address the information types needed, per information type.

type of information	# reports where it was not described	% reports where it was not described	# different reporters
What were you trying to do?	2	4.4%	2
Why do you want to do it?	**7**	**15.5%**	**4**
What did you do?	3	6.7%	2
What happened?	0	0	0
What were your expectations?	1	2.2%	1
What are you running?	**7**	**15.5%**	**7**

One of the purposes of this study was to assess to what extent, according to the findings reported on section 6.2, the form was successful in eliciting the information needed by OSS designers. Since the questions in the form were designed to map to the types of information we established previously (Table 10), we analyzed each of the collected reports, observing if the answers to each question provided the expected information or not. In the case the answer did

not address our expectations, we tried to find the needed piece of information in the answers to the other questions. The results of this analysis are presented in Table 16.

It is possible to observe that, using our proposed form, it was hard to elicit information related to the *why do want to do it?* information type. In the seven reports in which this information was not observed, the reporter rephrased or just repeated his answer to the question associated to the *what were you trying to do?* information type. In the comments section of the NASA-TLX form, three participants observed that the question related to *why do you want to do it?* felt "weird" or "silly" to answer. One of them said: "I see the need for the question, but I don't think it makes sense for all kinds of problem". Indeed, we observed that, for certain issues, *why do you want to do it?* might not be relevant to the problem being described. For example, one of the collected reports described a situation where the reporter could not understand an error message that showed up when he was trying to connect to the Internet. In this case, the reporter just answered "I wanted to connect to the internet" to both *what were you trying to do?* and *why did you want to do it?*. The reasons why this reporter wanted to connect to the internet might not be as relevant to the issue being described as the circumstances under which the issue occurred (for example, device being used, connection type, etc.). In that sense, the phrasing of the *why do you want to do it?* information type might not be adequate to elicit certain kinds of context information. Additionally, we believe that, in a redesign of the form,

the question related to *why do you want to do it?* should be made optional. The form would then have to be redesigned to communicate when this information might be useful, so that the reporter can decide when to provide it or not.

Through our analysis, we also observed a number of problems with the question related to the *what are you running?* information type. We noticed two different scenarios for that. In the first one, the reporter omits version information, just mentioning the name of the software used. In that case, reporters often provided answers such as "The latest version" or "I don't know the version". Indeed, one of the participants commented that finding out version information was the worst part of using the form. He said: "I have to actually open the software and figure out the version (which might not be always easy to do). It would be better to have some sort of automatic version detection for this". In the second scenario, the reporter attributes the failure to another piece of software, not mentioning the one responsible for the issue. This was especially observed in reports describing issues with Web applications, in which the reporters provide version information related to the browser and not to the Web application. We believe this scenario might be due to actual misattribution of the issue, but also to the lack of version information in Web applications. As a solution to the latter, the question related to *what are you running?* might be rephrased to include other ways to identify a software's state when the reported issue was experienced (such as date of use and not only version, for example).

We also observed that only two out of the 45 reports presented symptoms of negative or offensive attitude from reporters, one raising questions with regards to the designer's skills and the other with generic dislike statements towards the new version of a software, such as “[previous version] was much better” or “it is just complicated and messy”. This result, however, might be biased due to the fact that participants knew that the information they provided would be evaluated (Adair, 1984).

Additionally, we believe that the form positively influenced the reporting practices of the four participants who also participated in our analysis of bug reports. All seven reports submitted by P1, P2 and P4 using our form were complete in terms of the information needs of OSS designers. In contrast, their reports in the GNOME bug tracker presented different symptoms of misalignment to the information needs of OSS designers. Out of the 25 bug reports from these participants that we analyzed, only four (all from P2, from which we analyzed 18 bug reports) were considered complete. It is worth observing that, from these 25 bug reports, three (one from P1, and two from P2) were not considered for comparison, since they reported cosmetic fixes to the interface, such as buttons alignment.

When it comes to P3, however, two of his three reports submitted through our form were incomplete, specifically in terms of the *what did you do?* information type. In the GNOME bug tracker, we analyzed eight bug reports from P3, four of them also missing information related to *what did you do?*.

Seven out of the eight bug reports presented some symptom of misalignment to the information needs of OSS designers, one common symptom being the absence of information related to *what were you trying to do?* and *why did you want to do it?* (three reports). This was observed in ten out of the 25 bug reports in our sample from P1, P2, P3 and P4.

Nature of reported problems

When accessing our form, reporters were prompted to choose one out of four problem descriptors to characterize the HCI issue they wanted to report: *I don't like the way something works, I can't figure out how to do something*, *Something is confusing or unclear*, and *Other*. As explained in section 7.1, these descriptors were designed to speak for different sets of user utterances, the questions in the form varying depending on the descriptor selected by the reporter.

In order to evaluate the descriptors and the way the user utterances were grouped, we tagged the collected reports using the set of user utterances considered when designing our form (Appendix B). Having this in mind and our grouping of the user utterances based on descriptors, we identified the descriptor we expected the reporter to select, and compared it with the descriptor they actually selected. In the 45 reports collected, 16 (35.6%) presented divergence between the descriptor we expected and the descriptor selected. Five out of 45 reports were tagged with two descriptors at the same time, since they reported issues tagged with user utterances grouped under

distinct descriptors. In none of them, the descriptor selected by the reporter was different from both the descriptors we used. These cases were not considered as divergences.

Table 18: Distribution of bug reports per descriptor expected and used.

selected \ **expected**	I don't like the way something works	I can't figure out how to do something	Something is confusing or unclear	Other	
I don't like the way something works	13	1	8	1	**23**
I can't figure out how to do something		3		1	**4**
Something is confusing or unclear	1		12		**13**
Other		2	2	1	**5**
	14	**6**	**22**	**3**	

The distribution of reports per descriptor expected and used is presented in Table 17. We hypothesize that the high number of occurrences of the *I don't like the way something works* descriptor, in comparison to what we expected, might be due to the question phrasing and to the fact that it always came first in the list of descriptors offered to reporters. For example, it could be thought that *I don't like the way something works* because *Something is confusing or unclear*, or because *I can't figure out how to do something*. In that sense, the ordering of the options and the lack of specificity of the *I don't like the way something works*

descriptor might have influenced our results.

Table 19: Occurrences of each user utterance per affordance level.

utterance	affordance levels in which utterance may occur according to Silveira and colleagues (2004)	operational	tactical	strategic	total
What happened?	operational, tactical	8	13		21
How do I do this?	tactical		6		6
Is there another way of doing this?	strategic			6	6
Help!	all	0	5	0	5
Where is it?	operational, tactical	3	1		4
Thanks, but, no thanks.	strategic			4	4
Why doesn't it?	Tactical, strategic		3	0	3
Why should I do this?	strategic			3	3
Oops!	operational	3			2
Where am I?	operational	1			1
What's this?	Operational, tactical	0	1		1
On whom does this depend?	operational	1			1
I give up.	tactical		1		1
Who can do this?	operational	1			1
I can't do it this way.	Tactical, strategic		0	1	1
What is it for?	strategic			1	1
		16	**30**	**15**	**61**

Tagging the reports with our set of user utterances also revealed other interesting aspects of our collected data. In our analysis, given the fact that

according to Silveira and colleagues (2004) some utterances might occur at more than one affordance level, we tagged the reports using the user utterance observed and the affordance level in which that user utterance occurred.

Table 18 presents the number of occurrences of each user utterances per affordance level, and shows only the utterances that were observed in the collected reports. Gray slots represent cases where a utterance conceptually can't occur in a certain affordance level, while white slots represent cases where the occurrence is possible but wasn't observed. It is possible to observe that the majority of the problems reported through our form are related to utterances occurring at tactical level, followed by utterances at operational level and, then, at strategic level. Further research is needed to identify either the form motivated the reporting of this kind of issues, or if it just allowed us to better recognize problems related to higher affordance levels.

We also observed that the four participants that also filed bug reports to the GNOME bug tracker proportionally reported more issues at higher affordance levels than they did at the bug tracker. In the nine reports volunteered by P1, P2, P3 and P4, we observed the occurrence of 3 utterances at operational level, 7 at tactical level, and one at strategic level. In the 33 bug reports in the GNOME bug tracker from these same participants, we observed the occurrence of 10 utterances at operational level, 5 at tactical level and none at strategic level. It's worth noting that some reports and bug reports had more than one utterance tagged to them, and some had none.

From Table 18, it is also possible to observe five occurrences of the *Help!* user utterance, even though it was not included in the set of utterances we used to design the form. In the reports tagged with this utterance, reporters mentioned using web search engines to look for instructions on how to achieve something, or resorting to instructions provided by help systems or other documentation materials. The latter case occurred in two reports in which the reporters described looking for instructions on how to install something.

In addition to what we exposed so far, we observed that nine of the 45 reports described situations where a piece of software crashed, froze or stopped working somehow. From these nine reports, seven also mentioned problems understanding the reason why the software failed, or how to avoid the failure to happen again. Apart from that, three reports described performance issues, situations where the system's response was not immediate and there was no indication of whether the user's action was ineffective, the software froze, or the system was working on a response.

We also observed five reports describing situations where the reporter had issues figuring out how to do something they used to do with an older version of a software, using a newer one. Another kind of report that we observed, with six occurrences, was related to features that do not exist in a certain piece of software. We noticed that, in the reports composed using our form, the rationale and motivation behind the feature suggestion was clear.

Workload assessment with NASA-TLX

In order to calculate the workload of filling our form, we followed NASA-TLX's instructions to compute weighted workload scores for participants. Figure 2 shows the overall workload for each of the participants in our study. In a scale from 0 to 100, the average workload for the task of filling our form was 24.49, with standard deviation of 16.68.

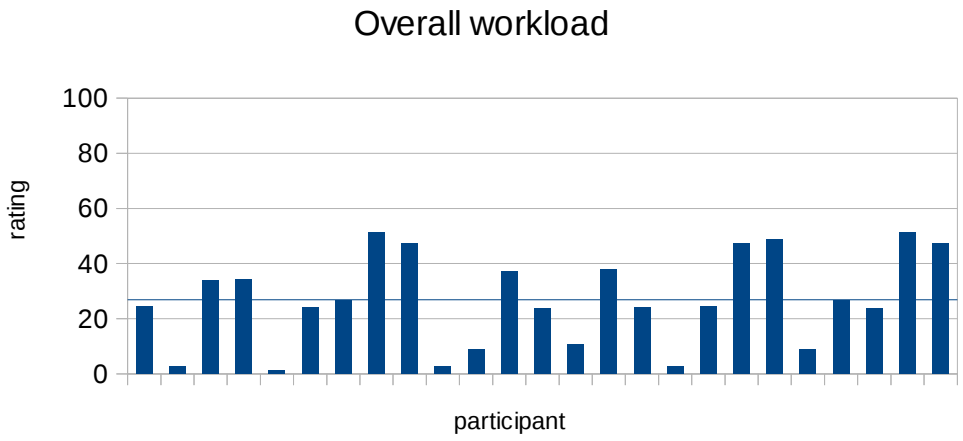

Figure 2: Overall workload for each of the participants in the study.

The NASA-TLX is a two part evaluation procedure consisting of both weights and ratings. Weights correspond to how a participant evaluates the contribution of each of the six dimensions (presented on section 7.2.1) to the workload of a task. Ratings correspond to the magnitude of each of the dimensions to a task. As defined in the procedure, the workload of a task for a participant is the weighted average of this participant's ratings for the six dimensions.

Table 19 displays the average, standard deviation (σ) and coefficient of variation (CV) values for the ratings of each of the six dimensions considered by

the NASA-TLX. As previously mentioned, ratings are given in a scale from 0 to 100. It is possible to observe that the ratings, especially for *Physical demand* and *Frustration*, present high values for CV, showing a big variation on how different participants perceive the magnitude of each dimension in relation to the task.

Table 20: Average rating per dimension, ordered by average rating.

dimension	**average rating**	***σ***	**CV**
Effort	31.8	21.9	0.69
Mental demand	26.4	20.9	0.8
Performance	25.4	24.6	0.97
Frustration	24.6	28.8	1.17
Temporal demand	17.0	17.5	1.03
Physical demand	15.4	24.4	1.58

As previously mentioned, weights are defined through paired comparisons between the dimensions. Paired comparisons require the participant to choose which dimension contributed more to the workload of the given task. The number of times a dimension is chosen as more relevant is, for that participant, the weight of that dimension for the task. Weights do not have values greater than five, since each dimension is compared against the other five dimensions. Table 20 displays the average, standard deviation (σ) and coefficient of variation (CV) values for the weights of each of the dimensions.

2122

Table 23: Average weight per dimension, ordered by average weight.

dimension	average weight	σ	CV
Performance	3.7	1.2	0.3
Effort	3.2	1.1	0.4
Temporal demand	2.8	1.6	0.6
Mental demand	2.3	1.4	0.6
Frustration	2.0	1.7	0.8
Physical demand	0.9	1.2	1.3

It is possible to observe that *Performance* and *Effort*, the two dimensions related to how a participant feels about the outcome of the task being evaluated, were perceived as the dimensions that contribute the most to the workload of filling the form we designed. This shows that, for the participants, having confidence in the quality of the composed reports plays an important role on the task of filling the form. Indeed, in the comments section of the NASA-TLX form, two participants mentioned not being confident that their reports were "correct" or would be understood.

We believe that, in order to address this and reduce the workload imposed

by our form, mechanisms for assuring the reporters of the quality of their answers are needed. This kind of mechanism could be inserted before or after the submission of a report. We also believe that having reports publicly available, as bug reports currently are, poses an opportunity for reporters to learn more about how to report HCI issues.

Overall, we believe the results of the NASA-TLX procedure were positive, and five participants mentioned that the form was objective and easy to understand. Two of them also observed that the examples following the questions were very helpful in terms of elucidating what was expected of them.

8 Conclusion

In this chapter, we discuss the contributions of this work and opportunities for further research.

8.1 Contributions

In this work, we discussed the importance of reports of HCI issues to designers in OSS projects, and ways to support users in creating them in a way that they are feasible to be acted upon.

In order to understand how feedback through reports of HCI issues fit and influence their design activities in OSS projects, we interviewed four OSS designers and analyzed a set of 547 bug reports filed in the GNOME project's bug tracker under HCI-related keywords. The first contribution of this work was to show the importance of reports of HCI issues for designers in OSS projects, and the obstacles they face when dealing with them. We identified a mismatch between the information OSS designers need in order to address reported HCI issues, and the information that is usually provided by users in reports.

With these interviews, we also elicited the information needed by OSS

designers in order for them to act upon reported HCI issues. The second contribution of this work was then the identification of the types of information OSS designers need in reports of HCI issues (see Section 6.2).

Based on the elicited types of information and on Semiotic Engineering concepts, we have designed and implemented a form for reporting HCI issues according to OSS designers' information needs, which is the third contribution of this work. We conducted a study in which 26 participants were invited to report experienced HCI issues using the form we designed. We wanted to observe how the form influenced the contents of reports of HCI issues in terms of the types of information needed by OSS designers.

We collected a total of 45 reports of HCI issues experienced with different kinds of software. From this study, we observed that our form was successful in eliciting the needed information for 36 of the reports collected. Additionally, we observed a high occurrence of reports describing issues at higher affordance levels. These results reinforce the designed form as a contribution of this work. It was also noted, however, that the form could be improved in terms of eliciting information related to software version and to context of use. Also, from an analysis based on the NASA-TLX procedure, we observed that reporters' confidence in the quality of the created reports plays an important role in the task of filling our form, exposing an opportunity for improvement from that perspective.

8.2 Future Work

Based on the analysis of the data collected, we have formulated some questions that can guide further work on reports of HCI issues in OSS projects.

The participants in our evaluation of the form came from a variety of backgrounds and had different experiences with technology and software. When analyzing the collected reports, we observed that reporters had different reporting styles and presented different symptoms of misalignment between the information provided and the information expected. Not only that, but we also observed a high variation between how participants perceived the magnitude of the NASA-TLX's dimensions for the task of filling our form. We raise the following questions: How does a user's background and past experience influence the way HCI issues are reported? How can we support different profiles of users in the task of creating reports of HCI issues that align with OSS designers' needs?

The latter question is also related to another question we want to raise with this work. We observed that performance, meaning how confident a reporter feels about composed reports, plays an important role in the workload of reporting HCI issues through our form. We believe it is important to provide ways to evaluate and assure reporters of the quality (or lack of quality) of their reports. To that respect, what mechanisms could be employed to achieve this goal? Additionally, how can we take advantage of the collaborative model of OSS projects to leverage learning and mentoring of best practices for reporting

HCI issues?

Another opportunity for further research comes from what we consider one of the limitations of this work. We elicited our list of types of information needed by OSS designers from interviews with contributors to the Fedora and GNOME projects, both of which have solid HCI strategies in place. This might have introduced some bias to our data. We wonder whether projects with other levels of commitment to HCI activities might have different information needs in terms of HCI issues. We take this opportunity to also question how other aspects of OSS projects might influence these needs: Do the information needs of OSS designers vary according to a project's conventions, culture and etiquette, for example?

This work addressed one of the challenges OSS designers face when dealing with reports of HCI issues: the mismatch between the information OSS designers need and what is provided to them in reports. In Section 6.1, we enumerated a number of other obstacles related to this matter. One of them is related to the management of reports of HCI issues, observed as the most time-consuming activity of OSS designers. Our work contributes with ways to support reporters to contribute with better descriptions of the problems they experience. We believe this might already reduce the time needed to process reports of HCI issues, by providing better problem descriptions, so that contributors (reporters, designers and developers) can focus on the discussion of solutions. Other dimensions to this problem, however, are the amount of

reports submitted and how to coordinate work to address reported issues between developers and designers. We ask: How can we support OSS designers in filtering, prioritizing and discussing reports of HCI issues? How can OSS designers better coordinate fixes to HCI issues together with developers? In time, what are the differences between the information needs of OSS designers and OSS developers?

Another obstacle faced by OSS designers is related to negative and/or aggressive attitudes adopted by reporters when submitting reports of HCI issues. We wonder how to address this issue from an HCI perspective, leveraging more positive attitudes through the design of issue report mechanisms.

Our form also presents several opportunities for improvement, such as: better phrasing of questions associated to context of use information (the *why did you want to do it?* information type), better ways to identify the version or state of a software when an issue was experienced, and tools for capturing and uploading visual information on issues, such as screen shots and videos. Another possible improvement is the addition of tips and resources to explain why and how the questions asked are relevant to describing HCI issues. Besides that, we believe that our grouping of the user utterances might be improved, given the 35.6% rate of divergence between the descriptor we expected to be used and the descriptor that was actually selected by reporters.

Finally, our form was designed having in mind the specific needs and

characteristics of HCI design in OSS communities. Further research is needed to investigate its suitability to other communities involved in HCI activities.

9 References

Adair, J. G. (1984). The Hawthorne effect: A reconsideration of the methodological artifact. Journal of applied psychology, 69(2), 334.

Andreasen, M. S., Nielsen, H., Schrøder, S., & Stage, J. (2006). Usability in open source software development: opinions and practice. Information technology and control, 25(3A), 303-312.

Andreasen, M. S., Nielsen, H. V., Schrøder, S. O., & Stage, J. (2007). What happened to remote usability testing?: an empirical study of three methods. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 1405-1414). ACM.

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug?. In Proceedings of the 28th international conference on Software engineering (pp. 361-370). ACM.

Bach, P. M., & Adviser-Carroll, J. M. (2009). Supporting the user experience in free/libre/open source software development. Pennsylvania State University.

Bach, P. M., DeLine, R., & Carroll, J. M. (2009). 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 (pp. 985-994). ACM.

Bach, P. M., & Twidale, M. (2010). Involving reflective users in design. In Proceedings of the 28th international conference on Human factors in computing systems (pp. 2037-2040). ACM.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008). What makes a good bug report?. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 308-318). ACM.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. Qualitative research in psychology, 3(2), 77-101.

Breu, S., Premraj, R., Sillito, J., & Zimmermann, T. (2010). Information needs in bug reports: improving cooperation between developers and users. In Proceedings of the 2010 ACM conference on Computer supported cooperative work (pp. 301-310). ACM.

Canfora, G., & Cerulo, L. (2006). Supporting change request assignment in open source development. In Symposium on Applied Computing: Proceedings of the 2006 ACM symposium on Applied computing (Vol. 23, No. 27, pp. 1767-1772).

Castillo, J. C., Hartson, H. R., & Hix, D. (1998). Remote usability evaluation: can users report their own critical incidents?. In CHI 98 conference summary on Human factors in computing systems (pp. 253-254). ACM.

Chilana, P. K., Ko, A. J., & Wobbrock, J. O. (2010). Understanding expressions of unwanted behaviors in open bug reporting. In Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on (pp. 203-206). IEEE.

Chilana, P. K., Ko, A. J., Wobbrock, J. O., Grossman, T., & Fitzmaurice, G. (2011). Post-deployment usability: a survey of current practices. In Proceedings of the 2011 annual conference on Human factors in computing systems (pp. 2243-2246). ACM.

Cox, A. (1998). Cathedrals, bazaars and the town council. Available from: http://slashdot.org/features/98/10/13/1423253.shtml. Accessed 02 December, 2012.

Crowston, K., & Scozzi, B. (2004). Coordination practices for bug fixing within

FLOSS development teams. In Proceedings of the First International Workshop on Computer Supported Activity Coordination (CSAC 2004).

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

Čubranić, D. (2004). Automatic bug triage using text categorization. In In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering.

de Souza, C. S., Prates, R. O., & Barbosa, S. D. (1999). A method for evaluating software communicability. Monografias em Ciência da Computação. Departamento de Informática. PUC-RioInf, 1200, 11-99.

Souza, C. S. D., Prates, R. O., & Carey, T. (2000). Missing and declining affordances: are these appropriate concepts?. Journal of the Brazilian Computer Society, 7(1), 26-34.

de Souza, C. S. (2005a). The semiotic engineering of human-computer interaction. MIT press.

de Souza, C. S. (2005b). Semiotic engineering: bringing designers and users together at interaction time. Interacting with Computers, 17(3), 317-341.

de Souza, C. S., & Leitão, C. F. (2009). Semiotic engineering methods for scientific research in HCI. Synthesis Lectures on Human-Centered Informatics, 2(1), 1-122.

de Souza, C. S. (2012). Semiotics. In: Soegaard, Mads and Dam, Rikke Friis (eds.), The Encyclopedia of Human-Computer Interaction, 2nd Ed.. Aarhus, Denmark: The Interaction Design Foundation. Available online at http://www.interaction-design.org/encyclopedia/semiotics_and_human-computer_interaction.html. Accessed 17 February, 2013.

Ducheneaut, N. (2005). Socialization in an open source software community: A

socio-technical analysis. Computer Supported Cooperative Work (CSCW), 14(4), 323-368.

Duffy, M. (2010) Fedora: A Case Study of Design in a FLOSS Community. Available online at http://duffy.fedorapeople.org/presentations/chi%202010%20floss%20hci%20workshop/duffy-flossdesign6.pdf. Accessed 02 December, 2012.

Faaborg, A., & Schwartz, D. (2010). Using a Distributed Heuristic Evaluation to Improve the Usability of Open Source Software. In CHI'10: Proceedings of the 28th international conference on Human factors in computing systems.

Fischer, M., Pinzger, M., & Gall, H. (2003). Analyzing and relating bug report data for feature tracking. In Proceedings of the 10th Working Conference on Reverse Engineering (p. 90). IEEE Computer Society.

Frishberg, N., Dirks, A. M., Benson, C., Nickell, S., & Smith, S. (2002). Getting to know you: open source development meets usability. In CHI'02 extended abstracts on Human factors in computing systems (pp. 932-933). ACM.

Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987). The vocabulary problem in human-system communication. Communications of the ACM, 30(11), 964-971.

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

Gallivan, M. J. (2008). Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. Information Systems Journal, 11(4), 277-304.

Ghosh, R. A. (2005). Understanding free software developers: Findings from the FLOSS study. Perspectives on free and open source software, 23-46.

Hammontree, M., Weiler, P., & Nayak, N. (1994). Remote usability testing. Interactions, 1(3), 21-25.

Hart, S. G., & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. Human mental workload, 1(3), 139-183.

Hartson, H. R., Castillo, J. C., Kelso, J., & Neale, W. C. (1996). Remote evaluation: the network as an extension of the usability laboratory. In Proceedings of the SIGCHI conference on Human factors in computing systems: common ground (pp. 228-235). ACM.

Hartson, H. R., & Castillo, J. C. (1998). Remote evaluation for post-deployment usability improvement. In Proceedings of the working conference on Advanced visual interfaces (pp. 22-29). ACM.

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

Heller, F., Lichtschlag, L., Wittenhagen, M., Karrer, T., & Borchers, J. (2011). Me hates this: exploring different levels of user feedback for (usability) bug reporting. In Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems (pp. 1357-1362). ACM.

Herbsleb, J. D., & Kuwana, E. (1993). Preserving knowledge in design projects: What designers need to know. In Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems (pp. 7-14). ACM.

Herraiz, I., Robles, G., Amor, J. J., Romera, T., & González Barahona, J. M. (2006). The processes of joining in global distributed software projects. In Proceedings of the 2006 international workshop on Global software development for the practitioner (pp. 27-33). ACM.

Hollnagel, E. (1993). Human reliability analysis: Context and control (pp. 147-202). London: Academic Press.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. (1991). User interface evaluation in the real world: a comparison of four techniques. In Proceedings of

the SIGCHI conference on Human factors in computing systems: Reaching through technology (pp. 119-124). ACM.

Jeffries, R., & Desurvire, H. (1992). Usability testing vs. heuristic evaluation: was there a contest?. ACM SIGCHI Bulletin, 24(4), 39-41.

Jeffries, R. (1994). Usability problem reports: Helping evaluators communicate effectively with developers. In Usability inspection methods (pp. 273-294). John Wiley & Sons, Inc..

Jensen, C., & Scacchi, W. (2007). Role migration and advancement processes in ossd projects: A comparative case study. In Software Engineering, 2007. ICSE 2007. 29th International Conference on (pp. 364-374). IEEE.

John, B. E., & Packer, H. (1995). Learning and using the cognitive walkthrough method: a case study approach. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 429-436). ACM Press/Addison-Wesley Publishing Co..

Just, S., Premraj, R., & Zimmermann, T. (2008). Towards the next generation of bug tracking systems. In Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on (pp. 82-85). IEEE.

King, N. (2012). Doing template analysis. Qualitative Organizational Research: Core Methods and Current Challenges.

Ko, A. J., Myers, B. A., Coblenz, M. J., & Aung, H. H. (2006a). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. Software Engineering, IEEE Transactions on, 32(12), 971-987.

Ko, A. J., Myers, B. A., & Chau, D. H. (2006b). A linguistic analysis of how people describe software problems. In Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on (pp. 127-134). IEEE.

Ko, A. J., DeLine, R., & Venolia, G. (2007). Information needs in collocated software development teams. In Proceedings of the 29th international

conference on Software Engineering (pp. 344-353). IEEE Computer Society.

Ko, A. J., & Chilana, P. K. (2010). How power users help and hinder open bug reporting. In Proceedings of the 28th international conference on Human factors in computing systems (pp. 1665-1674). ACM.

Lavery, D., Cockton, G., & Atkinson, M. P. (1997). Comparison of evaluation methods using structured usability problem reports. Behaviour & Information Technology, 16(4-5), 246-266.

Li, Q., Heckman, R., Allen, E., Crowston, K., Eseryel, U. Y., Howison, J., & Wiggins, A. (2008). Asynchronous decision-making in distributed teams.

Mack, R., & Montaniz, F. (1994). Observing, predicting, and analyzing usability problems. In Usability inspection methods (pp. 295-339). John Wiley & Sons, Inc..

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(3), 309-346.

Molich, R., & Nielsen, J. (1990). Improving a human-computer dialogue. Communications of the ACM, 33(3), 338-348.

Nichols, D. M., Thomson, K., & Yeates, S. A. (2001). Usability and open-source software development. In Proceedings of the Symposium on Computer Human Interaction (pp. 49-54). ACM.

Nichols, D. M. (2003). I'd like to complain about this software.... Workshop And Trip Reports Social Issues. SIGCHI Bulletin, 12.

Nichols, D. M., & Twidale, M. B. (2003). The usability of open source software. First Monday, 8(1-6).

Nichols, D. M., McKay, D., & Twidale, M. B. (2003). Participatory Usability: supporting proactive users. In Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction (pp. 63-68).

ACM.

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

Nielsen, J., & Molich, R. (1990, March). Heuristic evaluation of user interfaces. In Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people (pp. 249-256). ACM.

Nielsen, J. (1992). The usability engineering life cycle. Computer, 25(3), 12-22.

Nielsen, J., & Hackos, J. T. (1993). Usability engineering (Vol. 125184069). San Diego: Academic press.

Nielsen, J. (1994a). Heuristic evaluation. Usability inspection methods, 24, 413.

Nielsen, J. (1994b). Enhancing the explanatory power of usability heuristics. In Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence (pp. 152-158). ACM.

Nielsen, J. (1995a). How to conduct a heuristic evaluation. Available online at http://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/. Accessed 17 February, 2012.

Nielsen, J. (1995b). Characteristics of usability problems found by heuristic evaluation. Available online at http://www.nngroup.com/articles/usability-problems-found-by-heuristic-evaluation/. Accessed 17 February, 2012.

Norman, D. A., & Draper, S. W. (1986). User centered system design; new perspectives on human-computer interaction. L. Erlbaum Associates Inc..

Raymond, E. (1998). The cathedral and the bazaar. Knowledge, Technology & Policy, 12(3), 23-49.

Raymond, E. S. (1999). The revenge of the hackers. Open Sources–Voices from the Open Source Revolution', O 'Reilly, Cambridge, MA, USA, 207-220.

Runeson, P., Alexandersson, M., & Nyholm, O. (2007). Detection of duplicate

defect reports using natural language processing. In Software Engineering, 2007. ICSE 2007. 29th International Conference on (pp. 499-510). IEEE.

Salgado, L. C., Bim, S. A., & de Souza, C. S. (2006). Comparação entre os métodos de avaliação de base cognitiva e semiótica. In Proceedings of VII Brazilian symposium on Human factors in computing systems (pp. 158-167). ACM.

Sandusky, R. J., Gasser, L., & Ripoche, G. (2004a). Bug report networks: Varieties, strategies, and impacts in a F/OSS development community. In Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004).

Sandusky, R. J., Gasser, L., & Ripoche, G. (2004b). How negotiation shapes coordination in distributed software problem management.

Sandusky, R. J., & Gasser, L. (2005). Negotiation and the coordination of information and activity in distributed software problem management. In Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work (pp. 187-196). ACM.

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. In Software, IEE Proceedings- (Vol. 149, No. 1, pp. 24-39). IET.

Schön, D. A. (1983). The reflective practitioner: How Professionals Think in Action (Vol. 1). Basic books.

Schwartz, D., & Gunn, A. (2009). Integrating user experience into free/libre open source software: CHI 2009 special interest group. In Proceedings of the 27th international conference extended abstracts on Human factors in computing systems (pp. 2739-2742). ACM.

Sillito, J., Murphy, G. C., & De Volder, K. (2006). Questions programmers ask during software evolution tasks. In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering (pp. 23-34). ACM.

Silveira, M. S., Barbosa, S. D. J., & Souza, C. S. D. (2004). Designing online help systems for reflective users. Journal of the Brazilian Computer Society, 9(3), 25-38.

Stallman, R. (1992). Why software should be free. Available online at http://www.gnu.org/philosophy/shouldbefree.html. Accessed 01 December, 2012.

Twidale, M. B., & Nichols, D. M. (2005). Exploring usability discussions in open source development. In System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on (pp. 198c-198c). IEEE.

Viorres, N., Xenofon, P., Stavrakis, M., Vlachogiannis, E., Koutsabasis, P., & Darzentas, J. (2007). Major HCI challenges for open source software adoption and development. Online Communities and Social Computing, 455-464.

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

Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. (2007). How long will it take to fix this bug?. In Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on (pp. 1-1). IEEE.

Wilson, C., & Coyne, K. P. (2001). The whiteboard: Tracking usability issues: to bug or not to bug?. interactions, 8(3), 15-19.

Appendix A Interview Script

1. What is your role as a contributor?
2. How would you describe your design activities? What about your redesign activities?
3. Do you evaluate your designs? If so, how?
4. Do you seek feedback from users after the deployment of a design? If so, how?
5. What kind of information are you looking for?
6. What makes a good report of an HCI issue? Can you give examples?
7. How do you obtain feedback?
8. What kind of information do you usually obtain?
9. Do you succeed in finding the information you were after? How often?
10. What are the major obstacles to getting the information you need?

11. How do you process obtained feedback?

12. What motivates you to work on a given reported issue?

13. How does the obtained feedback influence your design activities?

Appendix B Tagging Reports

We used the same procedure to tag both the bug reports obtained from the Bugzilla instance of the GNOME project (Section 5.2), and the reports collected during the evaluation of our form (Section 7.2).

We tagged both sets of reports using a set of tags based on the user utterances of CEM (de Souza *et al.*, 1999). In addition to that, for the purpose of this study, the set of reports from the Bugzilla instance of the GNOME project was also tagged using a set of tags based on the heuristics proposed by Nielsen (1994b) for the Heuristic Evaluation.

The procedure basically consisted of going through each report twice. The first reading focused on identifying the main interaction problem(s) being described by the user. The goal of the second reading was to highlight snippets of the report describing the symptoms associated with the different user utterances or with violations of the guidelines described by the heuristics. Once identified, those snippets were then tagged with the correspondent user utterance or heuristic spotted.

Some examples of this tagging procedure:

- Tagging a bug report with heuristics

When doing a search in the Overview, the items under RECENT ITEMS do not have a right-click menu or other means to select an action to perform[1]. Some types of files may have more than one appropriate action.

For example, if I was working on a file called "index.html" yesterday, and I wanted to quickly pull that up to edit it, the Overview would find it okay but the default action (double-click) of showing it in the web browser would be the only option[2]. Ideally I could right-click to select which application to use (such as Gedit) as is the case in Nautilus.

I also tried drag/drop the icon from RECENT ITEMS[3] to Gedit in the Overview and that doesn't do anything either.

1. Flexibility and efficiency of use
2. Error prevention
3. Flexibility and efficiency of use

- Tagging a bug report with user utterances

It's easy to lose sense of where one is[1] when toggling between multiple workspaces (like with alt tab).

If you alt tab between windows on workspace 1 and workspace 5, the animations indicate that the workspaces are adjacent. It's very confusing then when the user then moves 1 workspace up or down and finds that the the windows she expected aren't there[2]!

1. Where am I?

2. What happened?

3.

Tagging a report submitted through our form using user utterances

What is happening?

I wanted to enable mobile broadband in Ubuntu 12.04.

Why were you trying to achieve that?

I wanted to access the internet, so I plugged in my mobile broadband/GSM device. But there was no option of enabling mobile broadband in Network connections[1].

Could you tell me, step by step, how were you trying to do it?

Initially on plugging the device, the mobile broadband was enabled and it showed so[2] (with the mobile broadband option checked). It then asked me for a password which I could not remember that time and clicked on cancel button in the dialog prompt[3]. Following this the broadband connection was lost[4] and so was the "mobile broadband enabled" option. I tried to register my connection again[5], but that also did not assure that I open my mobile broadband connection at will[6]. I tried plugging the device from start many a times too[7].

What was the problem?

The process of enabling mobile broadband connection is quite complex (as found in some internet sources). The option in the Network Connection drop-down is visible sometimes and sometimes not. I tried some of them, and finally re-plugged my device and this time luckily it showed the option and I could click on it.

How did you expect to do it?

I wish a consistent option for enabling mobile broadband, or an automatic enabling when a device is plugged in, or at least some kind of indication if something is wrong[8] in the settings (since the device was blinking properly and showed all signs of proper plugging).

What version of the software are you using?

Ubuntu 12.04 LTS

1. Where is it?
2. Looks fine to me.
3. I can't do it this way.
4. What happened?
5. Why doesn't it?
6. What now?
7. Why doesn't it?
8. Looks fine to me.

Appendix C HCI Issue Report Form

What's happening?
[select box]

1. I don't like the way something works.
2. I can't figure out how to do something.
3. Something is confusing or unclear.
4. Other

If reporter selected *I don't like the way something works.*:

What were you trying to achieve?
[text field]
I wanted to temporarily block my boss on my chat client, Empathy.

Why were you trying to achieve that?
[text area]
Because my boss is always bugging me about work on my spare time, but I don't want to appear as invisible to my whole list.

Could you tell me, step by step, how were you trying to do it?
[text area]
I located my boss on my Contacts list and tried to right-click over his name. I inspected the options, but couldn't find anything suitable. Then I tried checking Empathy's main menu, where there was this "Contacts" sub-menu with a "Blocked Contacts" item. I selected it and it opened a dialog with a list of blocked contacts. It had a minus ("-") button, but no plus. I closed the dialog and tried opening a chat window with my boss. There, I clicked on the "Contact" menu and then, finally, selected "Block contact".

What was the problem?

[text area]

The process for blocking my boss was very complicated, specially because I want to unblock my boss whenever I get to the office and then block him again when I leave. Apart from that, there was no indication of whether a contact is blocked or not.

How did you expect to do it?

[text area]

I wish I could block/unblock contacts easily, like by right-clicking the contact on my list. Also, once I blocked a contact, I'd expect his avatar to be grayed out or overlayed by some icon indicating "this is a blocked contact".

What version of the software are you using?

[text field]

Empathy 3.6.

If reporter selected *I can't figure out how to do something.*:

What were you trying to achieve?

[text field]

I wanted to temporarily block my boss on my chat client, Empathy.

Why were you trying to achieve that?

[text area]

Because my boss is always bugging me about work on my spare time, but I don't want to appear as invisible to my whole list.

What went wrong?

[text area]

I can't figure out how to block contacts on Empathy.

Could you tell me, step by step, how were you trying to do it?

[text area]

I located my boss on my Contacts list and tried to right-click over his name. I inspected the options, but couldn't find anything suitable. I tried selecting the "Edit" option to see if there was anything useful there, but I couldn't find anything.

What version of the software are you using?

[text field]
Empathy 3.6.

If reporter selected *Something is confusing or unclear.* or *Other:*

What were you trying to achieve?
[text field]
I was trying to copy some videos to my pen drive.

Why were you trying to achieve that?
[text area]
So I could watch them on my friend's computer, since I cannot plug my computer to the TV, as it doesn't have an HDMI plug.

Could you tell me, step by step, how were you trying to do it?
[text area]
I opened my Downloads folder, where the videos were, selected them and then pressed Ctrl+C to copy. Then, I opened my pen drive's directory and pressed Ctrl+V to paste the videos there. I waited until the transfer was over and ejected my pen drive, by pressing the eject icon close to its name in the side bar.

What went wrong?
[text area]
I got a notification saying "Writing files to pen drive" or something like this, but was never told when it finished.

What did you expect to happen?
[text area]
I expected a clear sign that I could remove my pen drive without damaging my files, like a notification or something.

What version of the software are you using?
[text field]
GNOME 3.6 and the Nautilus file manager (3.6 too).