Usability Discussions in Open Source Development

Michael B. Twidale David M. Nichols

Working Paper 08/2004

Department of Computer Science
University of Waikato
Hamilton, New Zealand

Usability Discussions in Open Source Development

Michael B. Twidale[1] David M. Nichols[2]
[1]*Graduate School of Library and Information Science,*
University of Illinois at Urbana-Champaign, IL, USA
twidale@uiuc.edu
[2]*Department of Computer Science, University of Waikato,*
Hamilton, New Zealand
dmn@cs.waikato.ac.nz

Abstract

The public nature of discussion in open source projects provides a valuable resource for understanding the mechanisms of open source software development. In this paper we explore how open source projects address issues of usability. We examine bug reports of several projects to characterise how developers address and resolve issues concerning user interfaces and interaction design. We discuss how bug reporting and discussion systems can be improved to better support bug reporters and open source developers.

1. Introduction

In contrast to the strong claims for the power, flexibility and robustness of open source software (OSS), its usability has at times been considered relatively weak [7,15,17]. In previous work we have considered this issue and explored reasons why it may be so, and how it can be alleviated [14,15]. In this paper we describe the first stages of an analysis of discussions about usability issues within current OSS projects. Even though our work is preliminary, the wealth of data available allows us to sketch out some aspects of how distributed design teams currently discuss interface design problems and solutions, and some of the problems that they encounter. We note how the text-centric design of bug reporting tools such as Bugzilla impose challenges to discussions about dynamic interactions with graphical user interfaces, and explore some of different ways that people cope with this. From an examination of a sampling of interface bugs, we have examined the usability discussions and contrasted them with the idealizations of interface design proposed in textbooks and with our own experiences of interface design and testing. This has led us to consider how aspects of complexity management and available technologies affect interface design discussions.

In Section 2 we describe previous work on analysing bug repositories. We then describe our analysis of interface issues in the context of several open source projects and discuss how these issues are reported, discussed and resolved. In Section 5 we identify areas for further investigation and possible tool support.

2. Background

Research on OSS development has recently started to address the wealth of data available in public software repositories and bug databases, (e.g. [3,6,9,11,13,15,18,19,20]. The issue of bug management has received relatively little attention in the research community and detailed studies using ethnographic observational techniques, such as [5,21], are rare. The proprietary nature of much software undoubtedly leads to a reluctance on the part of software companies to allow access to their (potentially embarrassing) bugs. In contrast, open access to the records of OSS projects allows investigations to be performed on a range of issues, from high level functionality to the text on a particular button.

Crowston and Scozzi [6] observe that the literature doesn't contain much detail on the processes involved in distributed bug-fixing; recent work such as [6,9,19,20] provides a valuable basis for a detailed analysis. In this paper we follow this strand of work but concentrate our attention on issues of interfaces and usability.

Previous work [1,7,14,15,17,22] has suggested that issues involving usability and user interfaces are not easily dealt (compared to 'code' issues) within OSS projects. Indeed, Wilson & Coyne [23] debate whether usability bugs even belong in the same database as 'mainstream' functional bugs. Recent evidence suggests that some OSS projects are adopting hybrid techniques that combine commercial approaches with the responsiveness and community involvement typical of open source [2].
Although some recent work touches on how OSS user interface (UI) bugs are processed [18,19,20] there is little detail in the literature [6,15]. In this paper we work towards determining whether interface design issues in OSS are different from other design issues and clarifying previously presented hypotheses as to the nature of usability in OSS projects [15].

Scacchi [20] takes as a starting point the conventional software engineering (SE) approach to requirements capture and specification, and then examines four diverse OSS projects in the light of that approach. He finds that the projects do not use the preferred SE approach of formal specifications, but that they do use 'informalisms' as lightweight mechanisms to help orchestrate development activity. In a similar way, we can take the conventional approaches to user interface design and development, and resource constrained variants (e.g. [16]), and examine aspects of OSS interface design and development in their light.

Open source bug reports allow a fascinating insight into the software design process, and into the aspects of usability analysis and design that we focus on. For researchers interested in any aspect of software development, this creates great opportunities, not just to study usability in OSS as a potentially problematic issue that can benefit from analysis and recommendations, but also for informing interaction design in all kinds of software development. The openness of OSS development allows access to the design process that can normally only be achieved in proprietary development by detailed, time-consuming ethnographic observation. The potential value of studying OSS and usability extends beyond OSS projects to the general HCI research community - as a real world laboratory of the acceptance, process and discussion of usability.

3. Method

Several approaches to analyzing OSS development have taken a quantitative approach (e.g. [11,13]) and only recently has a more detailed qualitative methodology started to appear (e.g. [6,9,18,19,20]). We follow this second strand of a more ethnographic-style low-level investigation which we intend to complement studies such as [13]. We use three sources: the Greenstone mailing lists and the Bugzilla [4] instances at Mozilla and GNOME.

The work reported here is a preliminary investigation of usability bug reports undertaken to gain a sense of the scope of how discussions about usability are undertaken. At this stage we are not ready to do a quantitative analysis of the amounts of each kind of action, but rather we aim for a qualitative sense of the kinds of discourse present, in order to understand how interface analysis and design occurs. In order to guide this exploration we focus both on what we see and what we do not see, that is in some sense 'surprising' in the light of other practices in usability design, such as standard HCI research and text book presentations. This parallels the analysis of software engineering practice in OSS projects contrasted to the textbook norms undertaken by Scacchi [20] and Massey [12]. Such an approach leads us to ask questions such as:

- What is the nature of usability discussions in OSS projects?
- Is it different from what might be expected from the textbooks on how to do interface design?
- Is it different from commercial software design?
- What are the patterns of discourse and process that emerge within and across projects?

We are aware that in the bug reports we only have a partial record of the work done and the discussions that occur; as work also happens on mailing lists, face-to-face meetings, newsgroups, blogs, instant messaging sessions (and their logs) etc . [3].

Our sampling for this approach was extremely ad hoc. We were not at the stage to make undertaking a systematic random or theoretical sampling. The best analogy might be a usability analyst clicking around with a new application in order to get a rough sense of what it is about and what it offers before undertaking a more systematic investigation. One aim is to generate some hypotheses that can then be tested in a more systematic study; focusing on issues that are surprising, or seemingly counter-intuitive, and that will need stronger evidence to substantiate than those aspects that are widely expected.

Our exploration involved searches of bug databases for terms such as 'usability', 'human computer interaction', 'interface' etc. Bug reports that contained those words and were determined to be indeed about usability were investigated in more depth, with a particular focus on how usability discussions use different kinds of evidence to support their claims.

In the following sections we take one extended example and several fragments to illustrate our findings. The example happens to be the first bug we chose to examine in depth for this paper. Its richness reinforced our observation of the wealth of the data in OSS projects. The quantity and quality is almost overwhelming. We believe that by attending to a few bug reports in depth, we can gain a rich understanding of the process of usability analysis and interface design as currently practiced. Our aims are to inform a more systematic study and to lead to sggestions, system enhancements and process changes for OSS interface design.

4. Results

In this section we discuss some of the components of usability dialogues that we found and note some aspects of the nature of the bugs and the discussion process that may lead to a bias towards certain kinds of activity. We begin with a somewhat extended example of a single bug that connects with many other bugs as an illustration of how interface discussions have many complex interrelated components. The rest of the section looks at aspects of the causes and consequences of this complexity. In our examples we have removed the identification numbers of the bugs we quote and do not note the authors of comments. However, all the information is publicly available.

4.1 Extended Example: dialog box sizing

Bug A opens with the description:

```
The preferences dialog size should be fixed
at a smaller size (closer to 4.x), this
size should be specified in ems and the
dialog should be made non-resizable
(Preferences is a dialog, not a window).
```

Comment 1 includes a patch as an attachment
Comment 2 reviews it
Comment 3 approves the patch but notes:

```
I think the patch is ok, but this is going
to cause a world of hurt. ;-)
```

Comment 4 records the bug as being fixed, less than 26 hours after its filing.

Note how the bug description includes somewhat implicit usability design criteria that the current interface violates, that dialog boxes should not be resizable, unlike windows which should be. This is an example of the larger interface design criterion of maintaining internal (within-application) and external (between-application) design consistency [16], a norm that we found frequent reference to in bugs we examined. Using Scacchi's term [20], we would categorise this claim that dialog boxes should be non-resizable as a 'usability informalism'.

However the bug does not end after being marked fixed at comment 4. In comment 6 the rule (dialog boxes should not be resizable) is challenged on two counts: i) don't constrain the user unnecessarily, and ii) resizable windows are a useful workaround for some dialogue boxes where the content does not fit the available area:

```
I'm not sure we need to constrain the user
like this.  Maybe a minimum
size, but ? why a maximum?  Some content
doesn't fit in this area,
that's been an on-going problem, and
resizing the window -> larger
is a workaround.  reopening for
explaination.
```

This widens the discussion to the problem of non-fitting dialog boxes, including references to other bugs. The desirability of permitting the workaround to exist instead of forcing a redesign of all the problematic dialog panes is debated.

A new point is raised, relating to the redesign of preferences dialogues to save some space (thus partially addressing the text-overflow problem), acknowledging that this point is a separate bug and asking whether it is worth filing. The advantages and disadvantages of creating this bug are discussed. Next, the creation of meta bug B is noted:

```
to keep myself sane, i've created a meta bug to track pref panels whose content needs readjusting in order to fit: bug B . feel free to add bugs as blockers to B. thx!
```

The non-resizable dialog boxes rule is further challenged, by citing the existence of resizable dialog boxes. It is noted that the Linux window manager makes all dialog boxes resizable, rendering the issue moot in that context. Eventually the bug is labeled 'fixed' (for the third time) and verified, 5 days after the initial report.

In total, bug A has references to 8 other bugs, including meta bug B which tracks panels whose contents need to fit. Meta bug B lasts for 22 months, contains 58 comments and references to 17 bugs (none of which were mentioned in bug A) before being marked as a duplicate in favour of another meta bug: C. This bug, entitled "Ensure all Preferences panes fit entirely within the pane area using all bundled themes" was created 26 months ago and is still active at the time of writing, currently with 66 comments and references to 25 bugs in the comments and with 43 bugs marked as blocking it. In the light of this complexity, Bug A's comment 3 about 'a world of hurt' seems most apposite!

4.2 Talking about Interfaces

Our investigation of usability bug reports has concentrated on the nature of the discussion about usability, rather than say the way in which the work is coordinated [6]. Using Crowston & Scozzi's [6] classification for all bugs, we are concentrating on what happens within 'submitting', 'analyzing', 'fixing' and 'testing', and are paying far less attention to 'assigning' and 'closing'. Here we group the issues into just two broad categories: the initial reporting of the bug and the subsequent discussion about its analysis and possible solutions. We are interested in how the discussions of the use, meaning and redesign of graphical user interfaces takes place in projects, with an eye to considering how the process can be improved. For example, does the text-centric nature of tools such as Bugzilla impede innovative UI design, and if so, how can we help?

4.2.1 Initial Reporting

Some usability problems can easily be explained textually. For example, a bug entitled:

```
Suggest change "Close Other Tabs" to "Close
All Other Tabs"
```

contains as part of its description:

```
Help clearly show that this closes all
tabs other than the one that you're in.

However, my initial take on seeing it
was:
Oh Cool!  I can close some of these
other tabs!

I thought that perhaps it would have me
click on the ones I wanted to close.
```

However, not all usability bugs can easily be reported textually and graphics can be an invaluable supplement [10]. If not included in the initial report, developers can

```
 Bug reporter:
   Description of Problem:
   The  icon  for  the  volume  control  is
broken.

   Steps to reproduce the problem:
   1. compile gnome 2.2.0
   2. start gnome-panel

   Actual Results:
   Wrong icon

   Expected Results:
   the speaker icon

   How often does this happen?
   Always
```

```
 Bug reporter:
   Created an attachment
   screenshot  of  broken  volume  control
icon
```

```
 Developer1:
   Are  you  talking  about  the  icon  with
the little  red  "x"  in  the  upper right
panel?   Is   this   the   volume   control
applet?
```

```
 Bug reporter:
   the  icon  with  the  little  red  "x"  in
the  upper  right  panel  is  indeed  the
volume  control  applet  where  I'm  talking
about.
```

Figure 1. Clarification dialogue over a screenshot from `bugzilla.gnome.org`

request reporters to provide a screenshot in order to understand the issue:

```
sorry about the delay getting to this bug.
Are you still seeing this? If you are, can
you screenshot it? I'm afraid I don't quite
get the problem :/
```

An indication of the value of screenshots is that the GNOME Bugzilla has added a 'screenshot' keyword to help index bug reports. However, sometimes even a screenshot is not sufficient to uniquely identify the problem. Figure 1 shows a short dialogue from the GNOME Buzgilla where a textual description and a screenshot are not enough to locate the problem.

Figure 2 shows a screenshot attached to a bug report. The two notable features of this screenshot are the privacy blurring of text in the 'From' and 'Subject' lines of the Ximian *Evolution* application and the annotation of the precise area of concern for the reporter.

The blurring of text helps to maintain the privacy of the reporter's information. It is clearly the case that bug reports can reveal information about the reporter [14] and the reporter here has clearly gone to significant trouble (presumably in an external graphics program) to obscure the text whilst still reporting the bug. The graphical annotation in Figure 2 effectively pre-empts the type of clarificatory dialogue in Figure 1, to concentrate attention on the pertinent part of the interface. Figure 3 shows similar clarification is often required in the textual context of mailing list discussions and help requests.

4.2.2 Usability Bugs: Discussion & Solutions

After a bug has been reported, it is analyzed, often in terms of its underlying causes. Understanding why a user may be confused can help in designing a fix to minimize confusion. Sometimes there is a debate about whether what is reported is a bug at all, as we will consider later in the paper. The bug may need to be categorized in terms of its wider meaning and the appropriate level to consider it in the project. For example, in our extended example (Section 4.1), the individual problems with particular dialog panes may be considered manifestations of the broader design rule of all dialog panes fitting correctly. Meta bugs can help in creating a layer of abstraction to address levels of detail.

Accompanying the analysis, one or more candidate solutions are proposed and these are in turn analyzed, critiqued and refined, until a decision is made and one solution is implemented. Sometimes there are ripple effects from the bug identification and solution stages, which we explore later.

The discussion in the form of posted comments is inevitably text-centric. In some cases this is absolutely no problem because the initial bug and the various solution ideas revolve around the wording of interface elements (as in the ‘close other tabs’ example above where the problem, the solution and its debate were all handled

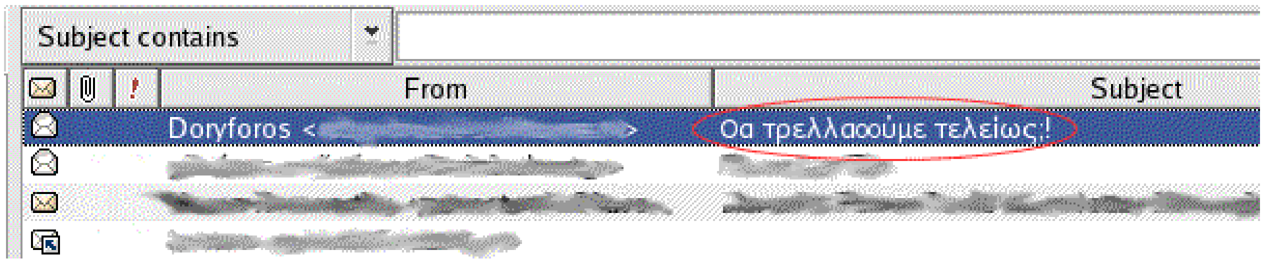

Figure 2. Privacy blurring and location annotation in a screenshot from **`bugzilla.gnome.org`**

User:

```
  Greenstone developers,
  I am facing a problem while putting my
html pages to the library.
  The collection is creating properly
and even the search is showing the
results, but when i click the search
result link it doesnt show the actual
content of the html page.
  I also tried the pages after placing
the files in the cgi-bin folder of
apache. Still this is not working.
  Any help or advice is greatly
appreciated. Thanks for your time and
consideration.
```

Developer:

```
  This is strange.
  A few questions:
  - Do you get an error message when
trying to view the documents, or just a
blank page?
  - Are you using Windows or Unix? If
Windows, are you using the local
library, or do you use a separate
webserver?
  - Have you made any changes to the
format statements in the collection
configuration file?
  - What version of Greenstone are you
using?
  If possible, the best thing would be
if you could send me a link to your
collection (so I can see the problem
myself), or, if the collection is small
enough, package the collection up and
send it to me. This would make it much
easier for me to track down the problem.
```

Figure 3 A report/help request and response from the `greenstone-users` mailing list

textually in a smooth manner). Rewording text elements of interfaces to improve usability seems to occur relatively frequently and we believe with good reason:

- Many usability problems can be addressed quickly and cheaply by rewording. It is far more cost effective to locate and change a text item in the code than do more substantial interface redesigns.
- In our experience of teaching usability, it is a context where relative usability novices can play a useful role, serving an apprenticeship before moving to more complex problems.
- Talking about the wording of interface elements is much easier to do in a mostly text based interaction medium such as Bugzilla than talking about graphical elements or interaction processes.

However non-textual resources are also used. We have already noted the power of screenshots and particularly annotated screenshots. We also find several examples of 'ASCII art' where some element of a UI are laid out within the textual discussion, see Figure 4.

The creation of ASCII art requires some effort, raising the question of whether functionality to provide more

```
 [Mozilla]
.---------------------.
| New               >-|-------------------------.
|---------------------|  | · Message             |
| Navigator           |  | · Address Book Card   |
| Mail & Newsgroups   |  .-----------------------.
| IRC Chat            |
| Composer            |
| Address Book        |
|---------------------|
| Preferences         |
|---------------------|
|*Apply Theme*      > |   *( It can be setup into
|---------------------|     \ 'Preferences'.I gues
| Work Offline        |      \ doesnt qualify enou
|---------------------|       \for have it out int
| Close               |        \ a menu too.)*
| Exit                |
.---------------------.

*If user wants to be open a new navigator window,
 just click 'Navigator'.
*May this menu should be named [Browser]
```

Figure 4. ASCII art from bugzilla.mozilla.org

lightweight ways to include design ideas would help. The presence of ASCII art has also led us to wonder about the relative lack of digitized freehand sketches, or other means of creating rough impressions of an interface idea. Indeed so far we have not found any examples of such sketches, although we feel sure that some must exist. Their relative rarity is particularly intriguing given the popularity of paper sketching and prototyping in the HCI literature [16], and the presence of quite polished interface specifications: carefully drawn faithful representations of what the interface should look like. The discussions about a bug and its candidate solutions draw on various kinds of evidence, we find references to usability studies, the HCI literature, usability concepts (such as Fitt's Law), and various guidelines and standards. Examples: (from several bugs)

```
Alerts should use OK, not Close.
http://developer.gnome.org/projects/gup/hig
/1.0/windows.html#alert-windows
or
http://developer.gnome.org/projects/gup/hig
/draft_hig/windows.html#alert-windows
This bug appears at least 17 times in
gnome-panel. The patch to follow
corrects those instances.
```

```
Does Apple, IBM or someone else provide any
good HIG's for context menus?
```

```
I just read an interesting chapter in a UI
Design book call "Designing for Both Sides
of the Screen"
```

Although we can find examples of these elements, we suspect that their use is relatively infrequent, especially in terms of the approaches and principles espoused in HCI texts about how to approach interface design. This needs more thorough investigation.

In examining a selection of usability bugs and the discussion about reporting, analysis and designing solutions, we have developed some concepts that we believe are helpful in making sense of what is currently done and why, and how approaches could be developed to support the interface design process. We explore two of those ideas in the following subsections.

4.3 Subjective and Objective Usability Bugs

The example in Section 4.1 contained a discussion of several bugs where elements did not fit within a 'Preferences' pane. Those individual bugs were relatively straightforward to report. The reporter noted the pane in question, and any actions that need to be taken to create the situation where the text or buttons do not fit the pane. A screenshot can be attached to illustrate the problem.

We would categorize this kind of usability bug as having the characteristics of being ‘objective’ and static. That is, once pointed out, most people would agree that it is indeed a problem, all users going through the same process would encounter this problem, and it can be considered a deviation from some standard, even if that standard is a set of informalisms and shared understandings about what should be rather than a deviation from a formal specification. The bug is static in the sense that it can be completely described using a single snapshot of the interface at a given point in an interaction, rather than requiring a consideration of an interaction sequence. In the case of dialog panes, a screenshot of text not fitting within the pane is sufficient to establish that there is a problem, most people would agree that indeed something should be done to fix the problem, and the only disagreement is about exactly how to fix the problem (in this particular example, actually there is substantial disagreement on the fix).

In our ad hoc sampling, we have encountered several examples of this kind of usability bug, and by contrast, relatively few ‘subjective’ usability bugs: aspects of an interface that cause confusion or errors for some people, but not everyone. An example of a subjective usability bug is the "Close Other Tabs" bug in 4.2.1.

We note this as being of interest for further investigation, because of a contrast with our experiences with usability testing of websites and interactive software. In user testing, one frequently encounters many subjective usability bugs, where one test user has a problem, but another does not. In some cases a single user error result is sufficient to persuade a designer that there is indeed a usability bug that needs fixing, but often other more convincing evidence is needed before a designer is convinced that there is indeed a problem, especially when it has no effect on others, who may be a majority. This variation in the use and hence usability of systems has frequently been used as a justification for frequent user testing, and as the reason that allowing developers to observe user tests has such a dramatic effect, showing designers how some people perceive their designs in a completely different manner to how it was intended [16]. In the absence of sensitivity to user testing, a developer on being told of such a subjective usability bug might well react “well it works for me”. The implication is that there is no failure of functionality and hence no bug. The fault is with the user who has failed to understand how to use the system properly, and the solution, if any, is with help or user training rather than system redesign, which would add more complexity and delay to the already complex design process.

It is worth noting that there is an explicit status within Bugzilla called ‘WORKSFORME’ used to note that a bug reported by one person has not been replicated by another. As such the option is a powerful tool for complexity management (see below), given that inevitably some bug reports will be erroneous, or need more precise details for replication. An unfortunate side effect of this highly desirable complexity management mechanism may be to generate a bias against usability bugs that don’t cause problems for expert users or that cannot be accepted as valid just by examining a visible failure in a single screenshot. These bugs may require an explanation of how some people taking some sequence through the use of the system are confused, make mistakes or are misled by the design of the interface, while others manage to cope, or even find the design helpful. More substantial evidence may be required before such a usability bug is regarded as legitimate – a single case is insufficient, while a formal usability test is powerful (but still contestable) evidence.

In its more extreme manifestations, a preference for bug objectivity may even lead to a downgrading of the perceived ‘status’ or acceptability of certain usability bugs. Objective usability bugs and functionality bugs may get greater attention because of a view that it is better to fix the problems that affect all users or lots of users (especially those that affect the OSS developers themselves), and are relatively uncontroversial in their existence, and only then to move on to those aspects that affect fewer people. It may be that the subjective bugs are treated more as enhancements, by analogy to additional pieces of functionality that will greatly benefit some but not all or even most users.

Examples.

```
Marking invalid, since this is as designed
and thus not a bug.
```

In this case a usability problem report is dismissed as being a legitimate bug since the issue reported as problematic performs exactly as intended. In effect this comment states that a usability bug is not a bug if the design was intentional. Note that we are not denying the legitimacy of marking a usability bug 'WONTFIX', due to the real or perceived rarity of the problem, or because all solutions involve undesirable trade-offs. Rather we question the dismissing of the very existence of a usability bug on the grounds that the functionality is not a deviation from the specification.

```
This is not a bug. There's an extension
called Tabbed Browsing Extensions that does
exactly what you want.
```

The comment above implies that because it is possible to download and integrate an extension to provide the enhancement that would address the usability problem described (a confusion about how tabbed browsers operate), there is in fact no bug, because it is possible to obtain an extension that changes the interface precisely in the way discussed in the analysis and fix discussion of the bug report. This is indisputable if attention is only paid to functionality, but if usability, particularly for novices is a legitimate issue of discourse, then such a comment should not simply dismiss the concern, given that the advocated solution to a novice's confusion with the default setup involves an even more confusing installation of an extension.

This issue needs more study, but one indicator is that a search over all sub-projects in Mozilla's Bugzilla for bug reports containing the word 'usability' in any comment yielded 1985 hits, of which 528 (27%) were in bugs marked as enhancements, even though enhancement bugs make up only 8% of all bugs in the database. (This is of course a crude indication – there may be many usability bugs that do not use the word 'usability', and many that do that are not really about a usability bug per se, but it does indicate that the issue is worth further investigation).

4.4 Complexity Management

In any open source project, the management of the complexity of the process is critical. Software development is inherently complex**.** Open source development has the additional complexity of large, heterogeneous distributed teams of volunteers. The way in which OSS manages to achieve successful development is a key issue in many studies of OSS [6, 18,19]. The bug reporting and fixing process has become streamlined with processes and norms, some just implicitly understood by OSS project members, some articulated in supporting documentation and some explicitly supported in the software developed to support the OSS design process, such as Bugzilla.

One aspect of complexity management is the attempt to create unique, clearly identified and categorised bugs. Duplicate bug reports add to complexity and so are undesirable. Reporters are strongly encouraged to search the database to see whether the bug they are reporting is already there. This can be non-trivial, as the 'vocabulary problem' [8] means that there can be many legitimate ways of describing the same bug. Thus duplicates inevitably occur, and effort is expended in quickly identifying and marking duplicates.

Similarly, it is desirable that the identity of bugs does not change. Each bug should be in its own report rather than having multi-bug reports, and a single report should not morph in meaning. Deviations from these norms do occur, but the norms appear well accepted, and for very good reasons.

In our initial explorations of usability bugs, we have noticed that those bugs we examined are often complex, with many comments, many duplicates and containing references to other bugs. Our extended example illustrates this, and is by no means exceptional. We wonder whether usability bugs show greater complexity than other bugs, and we plan to investigate this further.

In our earlier work in this area [15] we identified various desirable additional activities that could be added to an OSS project improve software usability. Many of these, such as informal and distributed user testing of current functionality or proposed solutions would directly address the more subjective and dynamic usability problems that seem to be under-reported and gain relatively little attention in current projects. In the light of our current analysis, we have become increasingly aware that any such additional features and processes would need to be accompanied by extremely careful complexity management. We believe that the best way to do this is to better understand existing complexity management practices and to propose mechanisms that fit within and enhance those practices.

The use of meta bugs such as B & C in section 4.1 allow related bugs to be collected and tracked. In the case of interface design, within-application consistency is a widely accepted design guideline. Hence when a report about a particular problem occurs with one part of the interface (in this case a dialog box pane), whatever solution is decided upon needs to be applied to all other related bugs. Furthermore, the solution may require consideration of non-buggy interface elements just to ensure that all elements maintain consistency after a fix. This means that the wider implications of local fixes have to be considered as alternate fix candidates are assessed, and then this additional ripple-effect fixing has to be monitored and achieved. The following two comments illustrate this

```
This has got to be a simple fix? It is in
1.0.1 on win98 too. If I knew how to fix I
would.
```

generating the response:

```
any single example would probably not too
much trouble to fix. but each panel needs
to be fixed to fit, on the full range of
```

```
platforms, fonts and resolutions, and as I
understand it, the various people
responsible for each of the pref settings
would need to be involved as well, to make
sure nothing got broken.  So while the code
changes are probably simple, there needs to
be a lot of people interaction, and that's
a hassle...
```

This nicely illustrates the complexity management problem in order to maintain interface design consistency. It can be contrasted with bugs relating to functionality, where for example changes in how a particular function works should need to only consider its callers, a set that is likely to be limited and easily identifiable. It appears that in interface design the complexity management technique of black boxing breaks down somewhat. One needs a 'T' shaped analysis: not just down through the particular interaction sequences addressed by the functionality associated with the fix for the buggy interface element, but also across all interface elements for consistency, including interface elements not functionally related to the reported bug.

```
please nominate specific bugs with specific
preference panels and not metabugs like
this one. thanks.
```

This comment articulates a complexity management norm, and it is clear that this norm makes a lot of sense when applied to functionality bugs. However with interface bugs and fixes, certain elements correlate so closely that it is desirable to advocate for all or none.

4.5.1 Ripple Effects of Bugs

Bug A is filed as an inter-application consistency violation; dialog boxes should not be resizable. In fixing this bug, it creates or accentuates other bugs; dialog boxes whose information no longer fits within the pane. Resizable dialog boxes had been used as a workaround for this problem, although one that various commentators to bug A saw as rather clumsy. The consequence was that fixing one bug created the need to fix other bugs.

We call this a ripple effect of bug fixing. Clearly it adds to complexity, and it would be highly desirable if rigorous modularization minimized or eliminated ripple effects. We speculate that interface bugs are particularly susceptible to ripple effects due to the impact that one interface element can have on a wider overall user experience. For example, a recurrent debate throughout the Mozilla project has been the size and complexity of the preferences dialog box. Frequently a particular interface or functionality bug leads to conflicting opinions about a suitable fix, sometimes leading to the creation of options, chosen via the preferences feature. Unfortunately, each additional preferences option although in itself minor, adds to the overall complexity of the preferences interface, potentially rendering it unusable to all but the most advanced users. This in turn may lead to a redesign, either hiding preferences or even removing options in order to improve usability, reduce the size of the code, and to improve testability, but at the expense of more limited choice.

In a similar manner, in our first example, as the redesign process continued through the meta bugs B and C, the initial bug A was continually challenged for legitimacy, and current versions of the software now do provide resizable dialog boxes.

5. Discussion

Despite the preliminary nature of our study, we believe that it is possible to identify certain issues that may inspire design solutions. These include: Scarcity of expertise, Bug reporting and classification, and Heterogeneity in usability discussions.

5.1 Scarcity of expertise

Scacchi [20] has shown similarities in the requirements specification process of four very diverse projects (ranging from games to astronomy). What are the similarities and differences in usability discussions between projects? We may find that the discourses on usability engineering within and between OSS projects parallel their historic equivalent in commercial software development. Systems initially developed for use by expert users were gradually adapted for use in more contexts and by people less willing to learn an unduly complex interface. Within commercial software development, usability professionals have often noted the importance of making the case for usability [16,23]. There is a competition for resources and attention within projects, and usability professionals often have to simultaneously establish the legitimacy of their work, as well as trying to integrate their analysis and design into ongoing functionality development.

Are similar issues seen in the open source record? In our small sample we do indeed see such advocacy. At times we also see frustration in trying to manage discussions between the two very different worldviews of expert developers and end user advocates. This is made harder when usability expertise is scarce in a project. Without a certain critical mass, it can be difficult to establish the legitimacy of usability arguments against countervailing expert-user functionality-centric claims. The usability advocate can feel outnumbered and give up the fight, or be crowded out in discussions, particularly OSS online discussions where ideally a consensus emerges, but where consensus-based interface design may not always be possible or even desirable.

With limited numbers and isolation, it is likely that the progress of usability in any given project will depend crucially on the approach (both in analysis and design, but also in rhetoric) of the few usability advocates involved. This will lead to greater inter-project diversity in this aspect than in functionality development, where there are many OSS participants with appropriate skills and interests. One might also expect differences due to the nature of the software being developed. For example a complex piece of functionality such as Apache needs to focus on power for expert users and hardly at all on inexperienced computer users, whereas a mail application is aimed at a far wider user base.

5.2 Bug Reporting & Classification

We have already noted how more integrated support for annotated screenshots (including privacy tools) can enhance the initial reporting of usability bugs. Reporting tools that automatically provide contextual metadata further reduce the effort required by bug reporters. Subjective usability bugs may need more of a provisional approval process after initial reporting, (especially if from a single anecdotal source), as more evidence is collected of the relative incidence and severity of the bug. use of suitable keywords would distinguish provisional subjective bugs from more objective established bugs. That would enable investigation to continue without adding undue complexity to the system, and avoid premature discarding of the partial evidence as insufficient to count as a bug.

Sandusky et al. [18] suggest that a duplicate identification tool would be a valuable addition to OSS projects. Our analysis supports this suggestion and we note that the more detailed the metadata from the bug report the more effective such a tool would be. Tools such as GNOME *Bug Buddy* and the Bugzilla *Helper* promote structured textual reports but the clarification dialogue shown in Figures 1 & 3 and in numerous bug discussions shows that metadata is more valuable. Bug metadata more directly supports querying and partitioning of the bug reports which should help to reduce duplicates and parallel bug discussions.

Classifying any bugs is a complex process, and contributes to the problem of duplication. It would seem that classifying usability bugs is at least as difficult as classifying functionality bugs. One approach to classifying usability bugs may be to use the structure of the user interface itself as a hierarchical classification system. That is, the menus, sub-menus and dialog boxes of the interface become nodes in the classification hierarchy of the bug repository, and so a preferences dialog box pane bug gets classified by navigating within a representation of the system interface.

5.3 Heterogeneity in usability discussion

If discussions about interfaces are particularly complex, the conventional bug report design may be insufficient. A linear temporal sequence of comments may be perfectly adequate for cases where there are relatively few comments or when the comments document a straightforward temporal workflow. An example might be: initial bug report, elaboration, confirmation, refinement of details, allocation of work, proposed fix, review, comments, revisions, further review and fix. In the case of some usability bugs, the process of describing and analyzing the bug may be more complex and more contested, with different kinds of evidence proposed and debated. Alternate design solutions may be reviewed and debated, drawing on multiple assessment criteria, and debates between alternate fixes revolving around different, potentially conflicting goals. Potential resolutions and tradeoffs may involve revised designs which are further reviewed, with a need to make decisions even when consensus is not reached.

For such complexity, a linear listing of comments may be insufficient to enable participants to keep track of all the elements of the discussion. A form of threading within a bug report may help in grouping discussion elements. Explicit representations of design arguments, trade-offs and rationales can help in clarifying the multi-aspect nature of the more complex discussions.

Sub-arguments may be elided from the larger discussion in alternate venues either within the discussion area or completely outside it (forums, newsgroups, blogs etc), but maintaining clear links to and from the main discussion. A separate designated design area may support design brainstorming and discussions with results and gists of overall discussions brought back to the larger discussion within the main bug report. Likewise an area for discussing evidence from user studies, anecdotal observations, personal experiences, references to the HCI literature, design guidelines and interface specifications may help in reconciling complex argument structures before returning a result, or a listing of contested points back to the main discussion. This is a kind of sub-classing – breaking a complex problem and argument structure into separate sub-parts that are worked on separately before being integrated. The challenge is to preserve the simplicity of the existing system for situations where it works well, while supporting the more complex design discussions, including those that did not at first seem to be complex.

6. Conclusion

As many researchers have noted, the openness of open source projects is a highly valuable resource in revealing much of the process of software development. We have chosen to focus on the process of usability analysis and interface design, based on a initial interest in the 'problem' of usability within some open source projects. By examining a small sample of usability bugs, we have found a rich set of resources used in discussing usability that can be used to inform a more systematic investigation of the data and inspire improved tool support.

References

[1] C. Benson, "Evaluations of GNOME Usability: Expanding the Appeal of GNOME", presented at the *Fourth Annual GNOME User and Developer European Conference (GUADEC),* 2003. at

http://developer.gnome.org/projects/gup/articles/guadec-2003/guadec-paper.html

[2] C. Benson, M. Muller-Prove and J. Mzourek, "Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans", *Extended Abstracts of the Conference on Human Factors and Computing Systems (CHI 2004).* New York, NY: ACM Press. 2004, pp. 1083-1084.

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

[4] *Bugzilla*, The Mozilla Organization. at http://www.bugzilla.org,

[5] P.H. Carstensen, C. Sorensen and T. Tuikka, "Lets Talk About Bugs!" *Scandinavian Journal of Information Systems* 7(1) 1995, pp. 33-54.

[6] K. Crowston, and B. Scozzi, "Coordination Practices for Bug Fixing within FLOSS Development Teams", *Proceedings of the First International Workshop on Computer Supported Activity Coordination (CSAC 2004)* Porto, Portugal, 2004.

[7] N. Frishberg, A.M. Dirks, C. Benson, S. Nickell, and S. Smith, "Getting to Know You: Open Source Development Meets Usability," *Extended Abstracts of the Conference on Human Factors in Computer Systems (CHI 2002)*. New York, NY: ACM Press, 2002, pp. 932-933.

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

[9] L. Gasser and G. Ripoche, "Distributed Collective Practices and F/OSS Problem Management: Perspective and Methods," presented at the *Conference on Cooperation, Innovation & Technologie, (CITE 2003)* University de Technologie de Troyes, France, 2003.

[10] H.R. Hartson and J.C. Castillo, "Remote Evaluation for Post-Deployment Usability Improvement", *Proceedings of the Conference on Advanced Visual Interfaces (AVI'98)*. New York, NY: ACM Press, 1998, pp. 22-29.

[11] J. Howison and K. Crowston, "The perils and pitfalls of mining SourceForge", *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004).*. Edinburgh, Scotland, UK. 2004, pp. 7-11.

[12] B. Massey, "Why OSS Folks Think SE Folks Are Clue-Impaired", *Proceedings of the 3rd Workshop on Open Source Software Engineering*, *International Conference on Software Engineering (ICSE'03)*. 2003, pp. 91-97.

[13] A. Mockus, R.T. Fielding and J. Herbsleb, "A case study of open source software development: the Apache server", *Proceedings of the 22nd International Conference on Software Engineering*, New York, NY: ACM Press, 2000, pp. 263-272.

[14] D.M. Nichols, D. McKay and M.B. Twidale, "Participatory Usability: supporting proactive users", *Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer Human Interaction - New Zealand Chapter (CHINZ'03)*, Dunedin, New Zealand: SIGCHI New Zealand. 2003, pp 63-68, 2003.

[15] D.M. Nichols and M.B. Twidale, "The Usability of Open Source Software", *First Monday* 8(1) 2003. http://firstmonday.org/issues/issue8_1/nichols/

[16] J. Nielsen, *Usability Engineering*, Boston: Academic Press. 1993.

[17] E.S. Raymond, "The revenge of the hackers," In M. Stone, S. Ockman, and C. DiBona (eds.). *Open Sources: Voices from the Open Source Revolution*. Sebastopol, CA.: O'Reilly & Associates. 1999, pp. 207-219

[18] R.J. Sandusky, L. Gasser and G. Ripoche, "Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community", *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004)*, Edinburgh, Scotland, UK. 2004, pp. 80-84.

[19] R.J. Sandusky, L. Gasser and G. Ripoche, "*How Negotiation Shapes Coordination in Distributed Software Problem Management*", SQA Project Memo UIUC 2004-07, GSLIS, UIUC, IL, USA. 2004.

[20] W. Scacchi, "Understanding the Requirements for Developing Open Source Software Systems", *IEE Proceedings — Software* 148(1) 2002, pp. 24-39.

[21] S.V. Shukla, "Hit squads & bug meisters: discovering new artifacts for the design of software supporting collaborative work", *SIGCHI Bulletin*, 30(2) 82-84.

[22] P. Trudelle, "Shall We Dance? Ten Lessons Learned from Netscape's Flirtation with Open Source UI Development," presented at the *Open Source Meets Usability Workshop, Conference on Human Factors in Computer Systems (CHI 2002)*, Minneapolis, MN, 2002. at http://www.iol.ie/~calum/chi2002/peter_trudelle.txt

[23] C. Wilson and K.P. Coyne, "Tracking usability issues: to bug or not to bug?" *interactions* 8(3) 2001, pp. 15-19.