

Usability Processes in Open Source Projects

David M Nichols & Michael B Twidale*

Department of Computer Science, University of Waikato,
Hamilton, New Zealand
dmn@cs.waikato.ac.nz

*Graduate School of Library and Information Science,
University of Illinois at Urbana-Champaign, IL, USA
twidale@uiuc.edu

Abstract

In this paper we explore how open source projects address issues of usability. We describe the mechanisms, techniques and technology used by open source communities to design and refine the interfaces to their programs. In particular we consider how these developers cope with their distributed community, lack of domain expertise, limited resources and separation from their users. We also 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, its usability has at times been considered relatively weak (Benson 2004, Raymond 1999). In previous work we have considered this issue and explored reasons why it may be so, and how it can be alleviated (Nichols *et al.* 2003, Nichols and Twidale 2003).

In this paper we describe how usability issues are addressed within open source projects, how techniques from the discipline of human-computer interaction are applied and how companies contribute to open source development to ensure the quality of resulting applications. We present examples from the Mozilla and GNOME projects to understand the resolution of usability issues at a fine-grain level, using a qualitative ethnographic approach in contrast to quantitative analyses of source code repositories.

In section 2 we give an outline of previous work on open source usability. In section 3 we outline the sources, methodological issues and examples of current practice. Section 4 describes the main findings and is followed by a discussion and a conclusion.

2. BACKGROUND

The growing prominence of open source software (OSS) over recent years has led researchers to examine how open sources processes differ from conventional software engineering. This body of research has examined many aspects of open source development including: motivation (Hars and Ou 2002, Hertel *et al.* 2003), source code repositories (Mockus *et al.* 2002), sponsorship and governance (O'Mahony 2003, Stewart *et al.* 2005) and requirements (Scacchi, 2002). A smaller body of work (e.g. Benson 2004, Benson *et al.* 2004, Nichols and Twidale 2003, Twidale and Nichols 2005) has focussed on issues of usability. This topic of research has been motivated by a belief that usability is an area of perceived weakness for OSS; although there are few direct comparisons with proprietary applications to make clear judgements (Nichols and Twidale 2003).

The main arguments for the proposition that usability is a potential area of concern for OSS can be summarised as:

- Developers are not typical end-users
- Usability experts do not get involved in OSS projects
- Open source projects lack the resources to undertake high quality usability work
- Interface (re)design may not be amenable to the same approaches as functionality (re)design

One of the strengths of open source development, and the academic collaboration that preceded it (Feller and Fitzgerald 2002, Weber 2004), has been that the developers and users of the software have been very similar groups. Raymond (1998) comments that 'Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.' This 'user as co-developer' approach has proven very successful for programming development tools and server-side applications (Nichols and Twidale 2003, pg 114: Feller and Fitzgerald 2002). This success is, at least partly, derived from the similarity of the experiences of the members of the developer community. When the user group is distinct from the developers these differences can easily generate usability problems: assumptions that work within one community don't cross over to the other. The entire field of user-centred design (e.g. Nielsen 1993) is a response to this gap between developers and users.

As OSS has moved to creating applications that 'normal' users have on their desktop then the developer-user gap becomes apparent and the OSS development methodology has been questioned (Frishberg *et al.* 2002, Nichols and Twidale 2003, Raymond 1999). The challenge of producing quality web browsers, office suites and graphics tools is qualitatively different from writing a good web server.

Open source projects that follow the archetype of Raymond's (1998) developers 'scratching an itch' typically have not included members with significant usability expertise (Nichols and Twidale 2003, Raymond 1999). Possible explanations include a shortage of usability engineers and differences in culture between the disciplines of programming and usability (Nichols and Twidale 2003). Even large open source projects have had problems in this area: one of the 'lessons learned' in the Mozilla project is to "ensure that UI [user interface] designers engage the Open Source community" (Trudelle 2002). One of the reasons behind the lack of human computer interaction (HCI) expertise in OSS projects lies in the volunteer nature of the community; in contrast, companies can

employ usability engineers. In the absence of usability expertise, ‘itch-based OSS design’ will not succeed in improving certain kinds of ease of use. If existing powerful systems are unfortunately hard to use by non-hackers, those same hackers will not perceive an itch that needs scratching (Nichols and Twidale 2003).

Many open source projects have developed without significant commercial input. Whilst this shows the possibility for quality software to be written outside a commercial environment it is clear that many projects are at a significant resource disadvantage with their proprietary rivals. Some aspects of usability work, such as user evaluations in dedicated usability laboratories, are out of reach for many projects. However, as open source has gained in prominence, companies have started to engage with OSS projects and have provided significant resources to some projects.

"company-driven OSS projects, such as OpenOffice.org, where the company’s usability staff can continue to specify, design and test the majority of UI components in a more traditional fashion."

Benson 2004

For example, Sun Microsystems has contributed to the GNOME project (German 2003) through laboratory-based usability tests (Smith *et al.* 2001) and the development of a set of Human Interface Guidelines (HIG) (Benson *et al.* 2004).

One approach for commercial involvement in interface development is simply to layer a proprietary interface over an open source base, e.g. Apple's OS X. A related approach is to create a custom partial ‘wrapper’ around one aspect of the OSS functionality that is widely perceived to be complex, such as installing Linux. However these projects are the exceptions, most are seriously constrained by lack of resources and usability expertise: "few [open source] projects yet meet the most basic usability requirement: a thorough understanding of their target audience" (Benson 2004).

The overall picture, then, is of a community which is only recently starting to interact with the concepts of user experience design and usability engineering which have been common elsewhere for over a decade (Nielsen 1993). The open source community brings characteristics which are unusual in the world of HCI: extreme geographical distribution, lack of expertise, a code-centric world-view, lack of resources and a culture which can be somewhat alien to interaction designers. In the rest of this paper we examine how these two cultures intersect.

3. Usability in Open Source Development

In this section we outline some of the main features of usability work in open source projects. These observations mainly derive from ethnographic observation of the Mozilla and GNOME projects over past three years. Although much research on open source uses quantitative measures (e.g. Herbsleb and Mockus, Howison and Crowston 2004, Mockus *et al.* 2002) here we follow an observational approach (e.g. Scacchi 2002, Sandusky *et al.* 2004) based around bug repositories, mailing lists and blogs.

Before examining the usability issues that arise in OSS development and what we can learn from observing those development processes, we wish to note that research in this area has potential to be of value to all kinds of software development, not just OSS. The proprietary nature of much software undoubtedly leads to reluctance on the part of

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.

Although our previous work (Nichols and Twidale 2003, Twidale and Nichols 2005) has considered the special circumstances of the OSS development process that can contribute to making usability a major focus of work and consequently a major area of excellence in OSS, it is hardly the case that usability is a resolved problem in closed source software projects. There are far too many examples and stories of poor commercially designed interfaces and commercial projects that fail to take usability engineering seriously or allocate sufficient resources.

The openness of OSS development allows researchers to see much of the software development process that is normally kept hidden within the walls of commercial buildings. It would be unfair to compare imperfect but public OSS processes with imagined but concealed commercial processes. Evidence from other projects over many years shows that real life constraints frequently enforce departures from the idealizations of software development textbooks (Austin 2001, Gilb 1989); this happens just as much in closed source development just as much as open source. Although, many OSS projects don't have the same commercial pressures to 'ship' a product by a particular date.

Below we discuss three main issues in current OSS usability: complexity management, user involvement in bug reporting and the use of explicit interface guidelines.

3.1 Complexity

Previous work (Twidale and Nichols 2005, Sandusky *et al.* 2004) has focussed on bug repositories (such as Bugzilla instances) and has shown that understanding even small networks of inter-related bugs is a complex analytical task. In studying usability bugs we found that usability bug reports and their consequent analyses and redesign discussions have the potential to be particularly complex and contentious (Twidale and Nichols 2005). We believe that this additional complexity and contentiousness, compared to many functionality bugs, may occur for a number of reasons:

- *The very existence of certain usability bugs may be viewed as subjective and unreliable*
Usability bugs may not be afforded the same status as functionality bugs, particularly those latter that lead to crashes. A particular interface element can be confusing or ambiguous to some people but not others – what we term 'subjective' usability bugs. These subjective bugs are in danger of being tagged with the Bugzilla status 'WORKSFORME', which is conventionally used to note that a functionality bug reported by one person can not be replicated by another. This is a useful approach for overall complexity management (handling un-replicable bugs), but does create a bias against usability bugs.
- *Usability bugs can cluster in complex ways*
Often a usability bug spotted in one context can have similar manifestations in other parts of the interface, and this is important to note if within-application consistency is to be maintained. Similarly a fix for one usability flaw can have implications elsewhere in the overall interface design. As a result multiple bugs

- can be related and meta-bugs are used to help manage the complexity of the bug network (Sandusky *et al.* 2004). However, these can lead to parallel discussions of the same topic in multiple locations.
- *Discussions need to involve complex design rationales, multiple, perhaps contradictory goals and design trade-offs*
Usability bug analysis and fix discussions can become long, and it can be hard to keep track of all their elements. Twidale and Nichols (2005) describe an example from Mozilla's preferences panels, where a proposed fix improves the consistency of one aspect of a dialog box, but has implications for other dialog boxes, as well as raising issues of general cross-platform flexibility. The discussion also touches on a wider OSS functionality-usability-consensus problem, namely how alternate design solutions are frequently resolved by providing both as alternate options and this in turn leading to ever more complex configuration options and interfaces. Alternate design solutions should be debated in the light of these multiple goals and trade-offs, but later discussions may focus on one trade-off issue and omit factors critical to the understanding of the rationale of prior design decisions.

Our assessment of the relative complexity of usability and functionality bugs, assuming a clear distinction can be made, is still qualitative in nature and obtaining quantitative evidence is still an issue for further research. As we describe later, that evidence in fact would need to be drawn from sources beyond project bug repositories.

3.2 Reporting

Given the complexity of usability analysis and design discussions, debates can get ignored or side-tracked. The current bug reporting software, such as Bugzilla, seems to be inadequate for supporting effective usability discussions:

Difficulties that a user may experience with a graphical user interface may not be easy to describe textually.

It is possible to include screenshots within Bugzilla reports but it is an extra burden both to create them and to read them, distracting from the flow of a sequence of textual messages. Furthermore, even a screenshot and accompanying text may be insufficient to disambiguate what is being talked about - Figure 1.

```

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 the GNOME Bugzilla (<http://bugzilla.gnome.org>)

Workarounds to this problem that we have observed include the use of annotated screenshots and the blurring of elements of a screenshot to maintain the privacy of the reporter's information. In Figure 2 the bug reporter has gone to considerable trouble to blur the 'From' and 'Subject' information from the image. Further, the reporter has annotated the image to indicate the location of the perceived problem. Many bug reports we have observed at both Mozilla and GNOME do not have this level of detail and consequently clarifying dialogues are frequently necessary.



Figure 2. Privacy blurring and problem location annotation in a screenshot attached to a bug report at the GNOME Bugzilla (<http://bugzilla.gnome.org>)

Some usability confusions may have more of a dynamic element

The confusion can be an aspect of an action sequence rather than being reducible to a talking about single menu or dialogue box. In such cases even a single screenshot may be insufficient, no matter how easy to create or use.

Proposed design solutions can often best be explained and justified using non-textual means.

This might involve creating mock-ups in HTML, an interface design toolkit such as *Glade*, a standard drawing package, simple sketches, or even, and to us most surprising, the use of ASCII art, as shown in Figure 3. The presence of ASCII art is a good indicator that even in textual environments the nature of interface design requires graphical communication.

```
[Mozilla]
-----
| New          >-|-----
|-----| | . Message |
| Navigator   | | . Address Book Card |
| Mail & Newsgroups | |-----
| IRC Chat    |
| Composer    |
| Address Book|
|-----|
| Preferences |
|-----|
|*Apply Theme* > | * ( It can be setup into
|-----| | \ 'Preferences'.I guess
| Work Offline | | \ doesnt qualify enough
|-----| | \ for have it out into
| 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 3. ASCII art from a bug report in the Mozilla Bugzilla (<http://bugzilla.mozilla.org>)

3.3 GNOME Human Interface Guidelines

The Human Interface Guidelines are a significant contribution to the GNOME project as they provide a central reference point for much usability work (Benson *et al.* 2004). Figure 4 shows an example of a bug report referring to the HIG. The HIG can be an effective means to pre-empt a long discussion as they act as a source of authority on what changes *should* be made.

Bug 120716: "Remember logon" dialog is not HIG compliant

Reporter: 

The "Remember logon" dialog has the opposite button order of what the HIG requires. The order should be "Never for this site", "No", "Yes".

Ideally the button labels should be imperative verbs, I suggest "Never remember for this site" (this might be a bit long), "Don't remember", "Remember".

Reference:

<http://developer.gnome.org/projects/gup/hig/1.0/windows.html#alert-button-order>

Figure 4 A bug report using the Human Interface Guidelines (HIG) from the GNOME Bugzilla (<http://bugzilla.gnome.org>)

Yamauchi *et al.* (2000) describe the open source projects they observed as engaging in a 'rational culture; 'members try to make their behavior logically plausible and technologically superior options are always chosen in decision-making.' The use of the HIG in GNOME is a concrete example of this 'rational culture': it provides explicit (presumably best practice) guidance on issues of interface design. The prescriptive nature of the HIG also enables the community to orient itself towards 'action' (Yamauchi *et al.* 2000); the dialog should be re-designed *this* way.

The examples above illustrate the additional difficulties encountered in undertaking usability practices in open source software development. In the next section we attempt to systematise those difficulties and use that as a framework for understanding existing practice and possible future improvements.

4. ENHANCING REPORTING, ANALYSIS AND (RE)DESIGN

Given that there are difficulties with addressing usability, in this section we attempt to explore how reporting, analysis and design work occurs in current open source projects. By an analysis of the components of usability processes and their current manifestations as practices, workarounds and appropriations of available technologies, we aim to suggest mechanisms to improve current practice.

4.1 Improving Bug Reporting

We can envisage requirement elements of an improved system for supporting the handling of usability bugs that addresses the issues outlined in section 3. Knowing that there is (or may be) a problem with the system is of course critical to the success of any project, open or closed and applicable to functionality or usability bugs alike. Raymond (1998) proposes that "given enough eyeballs, all bugs are shallow", but of course with usability bugs, it can be critical to have the right kind of eyeballs. That means involving usability experts, but also end users who may be willing to note their confusion or frustration. Participation in bug reporting is acknowledged to be a tiny percentage of all OSS users. Nakakoji *et al.* (2002) notes that "generally speaking most members [of an open source community] are Passive Members. For example about 99 percent of people who use Apache are Passive Users". Similarly a user on the Mozilla Bugzilla comments:

Reports from lots of users is unusual too; my usual rule of thumb is that only 10% of users have any idea what newsgroups are (and most of them lurk 90% of the time), and that much less than 1% of even mozilla users ever file a bug. That would mean we don't really ever hear from 90% of users, unless we make some effort to reach them."

http://bugzilla.mozilla.org/show_bug.cgi?id=89907#c14.

Widening the participation in bug reporting will require lowering cultural, technical and usability barriers. Given the complexity and subjective nature of usability bugs and the need for wider participation as compared to addressing conventional functionality bugs, we suspect that this usability system should be distinct from a functionality-centric system such as Bugzilla. We believe that a usability bug infrastructure should contain features that can address issues of detection and consolidation.

Detection

To increase the likelihood of end users and usability experts reporting usability problems, we need a way of reporting usability bugs that is substantially easier than Bugzilla. Bugzilla requires registration and a careful following of a complex set of norms in order to ensure that bugs are reported with sufficient information to ensure replication and to minimise duplication. This imposes a significant burden on end users, and indeed is not necessarily ideal even for current users and uses of Bugzilla. Elsewhere (Nichols *et al.* 2003) we have considered ways in which crash reporting techniques could be adapted and extended to usability bug reporting. These could vary in the degree of participation required of a reported, from a very basic 'I'm confused' accompanied by an automatic determination of system state at the moment of declared confusion, through to a user-provided description of the nature of the confusion. The key twin concerns for widening participation in reporting are ease of use (so that it does not distract too much from the regular use of the software), and addressing privacy concerns about automated state data (so that the user knows that the information about their confusion does not include any confidential information).

Consolidation

Given the aim to widen the pool of potential bug reporters, any tool will need to deal with additional challenges: coping with (hopefully) a very large number of reports - including many duplicates, coping with erroneous, ambiguous and unhelpful reports, and classifying and consolidating these reports. Large numbers can help designers to get a sense of the relative priorities of which bugs to fix first, a critical piece of information given limited resources. In this sense, duplication of usability bugs (once consolidated) is valuable, whereas duplication of functionality bugs is often considered a waste of time and effort. Bugzilla addresses consolidation by expecting bug reporters to carefully search to see if their bug already exists in the database, and only if so, to add their report to it, provided it adds to the overall understanding of the problem.

The complexity of discussions in the Mozilla Bugzilla has led to the development of a new tool, *Hendrix* (Hendrix, 2004). Hendrix is an experimental and lightweight means for users to provide feedback for cases where the use of Bugzilla would be

onerous or inappropriate. It makes use of pre-existing newsgroups used to discuss possible extensions and bugs in Mozilla, separate from the formal Bugzilla reporting mechanism. Thus it separates out detection functionality and makes that even simpler for an end user to participate, whilst using pre-existing mechanisms for consolidation, analysis, design and consensus.

4.2 Analysis

Knowing that a particular interface element can cause confusion is not the same as knowing why. This can be achieved by careful analysis, by generating hypotheses, by comparing the interface to what is known about other systems or underlying HCI theory, by conducting user studies or by engaging the initial reporter in a more detailed discussion. In all cases analysis is likely to be discursive, needing to address multiple alternative candidate explanations and disagreements. Conventional bulletin board functionality is likely to be important, but supplemented by visual aids such as annotated screenshots to illustrate points under discussion. These must be sufficiently fast and easy to use that they do not get in the way of the discussion.

Some preliminary bug reports and analyses will be controversial. Developers may just not believe that a particular design element really does cause confusion, or for the reasons suggested, or that a proposed solution really is faster to learn or easier to use. In such circumstances, additional data might be needed, either by passively waiting for more bug reports to appear, or by actively undertaking usability tests. Remote user testing has been proposed as one possible approach (Hammontree 1994). Elsewhere (Nichols and Twidale 2003) we have advocated creating an infrastructure for very small scale usability studies, even as small as one user for 10 minutes. With a distributed team of volunteers and appropriate coordination, this can lead to aggregated results sufficient to inform ongoing development.

Design

Once the underlying cause(s) of the confusion has been determined by careful analysis, it is possible to propose design fixes to eliminate or ameliorate the problem. Of course care has to be taken that the proposed fix does not introduce new, worse problems, or cause a reversion to earlier problem that the former fix addressed. Team-based peer review, informed by a suitable design rationale of earlier and ongoing problems and goals can help here, just as they do in optimising OSS functionality and reliability. The additional complexity is the importance of consistency in interface design as a major element of overall usability. Hence local fixes have to be also considered for their larger global effects. In coding functionality, much effort is devoted to minimizing or eliminating global effects, and is a key point in software engineering. Unfortunately the global effects of interface consistency are critical and can not be eliminated.

Achieving consensus, (or at least making a coherent ruling)

There may be considerable disagreement about the relative importance or severity of a usability bug, whether it is a bug at all, its underlying causes, and the most appropriate design fix. Nevertheless, some consensus must be achieved if progress is to be made (or alternatively as we have found (Twidale and Nichols 2005), a failure to achieve consensus can leave a usability bug languishing unaddressed for years). The

importance of overall design consistency and look and feel, and a usability concern not to endlessly multiply options as a design resolution, may mean that if consensus is not possible a ruling must be made. One way to achieve this is to take the design work and discussion offline, enabling faster progress to be made. This can be done either in face-to-face meetings, or via synchronous technologies such as phone or instant messaging. In the absence of such mechanisms debates can just run on, or until one side runs out of enthusiasm for continuing the debate. In GNOME the HIG can provide an external source of authority to help a group of developers achieve a consensus; observations of usability bugs in the Mozilla Bugzilla suggest that the lack of a well-used set of interface guidelines may contribute to extended discussions.

4.3 (Re)Design Work

Although there is no specific usability design infrastructure in place analogous to the functionality-centric infrastructures such as CVS, SourceForge and Bugzilla, we do see informal uses that have the form of workarounds, appropriations and even the inspirations for 'informalisms' (Scacchi 2002) for a specification of a more usability-centric support structure. It is quite likely that such an infrastructure will evolve, just as one has already done with the development of tools such as Bugzilla. Such an evolution is likely to include components, or draw inspiration, from activities such as those we note below.

Moving design work elsewhere

We find mentions in the Bugzilla archive of design work, particularly interface design work, occurring 'elsewhere' than within the postings of Bugzilla. Possible reasons include:

- A desire to minimize the acknowledged complexity of Bugzilla by removing certain design debates until a solution can be devised that can be returned for public review and critique
- A wish to use Bugzilla solely for more formal review
- The remote asynchronous text-centric nature of Bugzilla is less than ideal for the brainstorming and design and debate of candidate solutions

Consequently, we find references to work occurring in face to face meetings, use of Mozilla newsgroups, IRC chats, emails, instant messaging etc. (von Hippel 2003). Each of these different venues in some way addresses one or more of the problems with Bugzilla for supporting interface analysis and design noted above. Unfortunately, although efficient, this can mean that the record of how the design evolved and how the decisions were made is lost, unfortunate both for researchers like us and more importantly for future redesign work.

4.4 Design-by-Blog: A new kind of off-loaded distributed design medium

One kind of technological appropriation we have found that addresses the problem of off-loading interface design from the conventional bug reporting and revision infrastructure, as embodied by Bugzilla, is the use of blogs. Blogs are used as

supplements to Bugzilla-like systems, rather than as replacements, for the reasons outlined above.

Many open source projects have blogs to disseminate news about their progress. Individual bloggers have many motivations for writing including 'providing commentary and opinions' and 'expressing deeply felt emotions' (Nardi *et al.* 2004) but from a project's perspective they represent any easy standards-based networked mechanism for maintaining communication across a distributed community. As the technology of blogs is largely content-neutral it has been a small step for projects to add to project news items with design and code-oriented posts.

Typically a design-oriented blog entry contains a problem description, information about the goals of the design and any constraints or design criteria, comments on earlier designs etc followed by screenshots of the proposed design. This in turn can be followed by text describing the design, its rationale and any particularly troubling or effective components. Further redesigns and rationales follow, often with a pattern of alternating screenshots and textual commentary and rationales. Additionally other people than the blog author will post comments on the evolving design. Of course, following the reverse chronological order of blogs, the evolving design has to be understood by reading *up* entries rather than down.

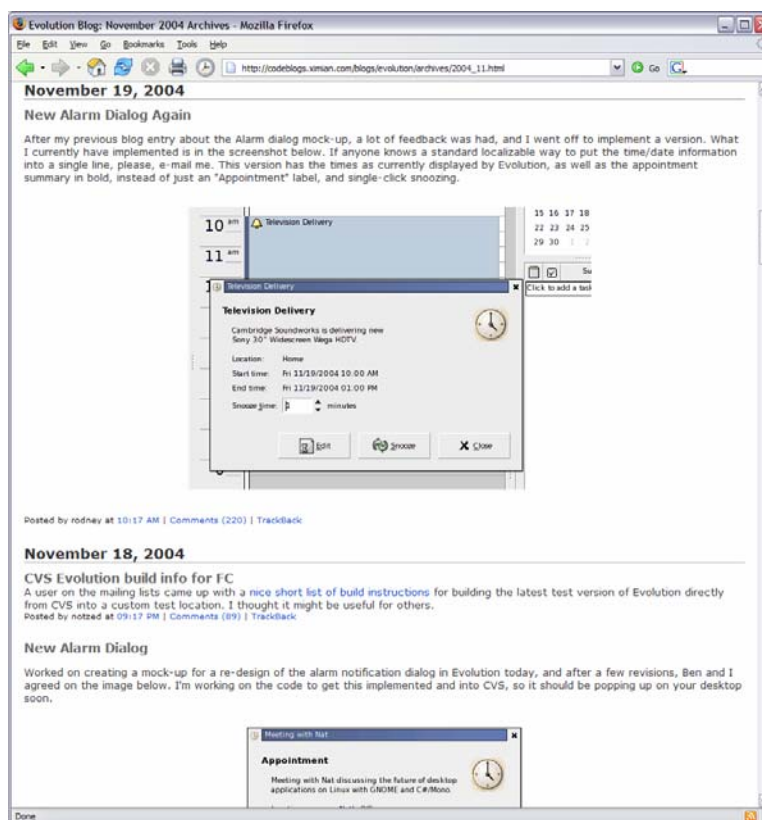


Figure 5. The *Evolution* Blog mixes project news with design suggestions

We illustrate how blog-based interface design can be collaborative, transparent and distributed with an extended example; Figure 5 shows a sample screenshot from the

blog associated with the groupware client *Evolution*. The blog mixes several types of information including project news, meeting minutes, requests for help and proposed design changes. Figure 6 shows three (re)designs of the alarm dialog.

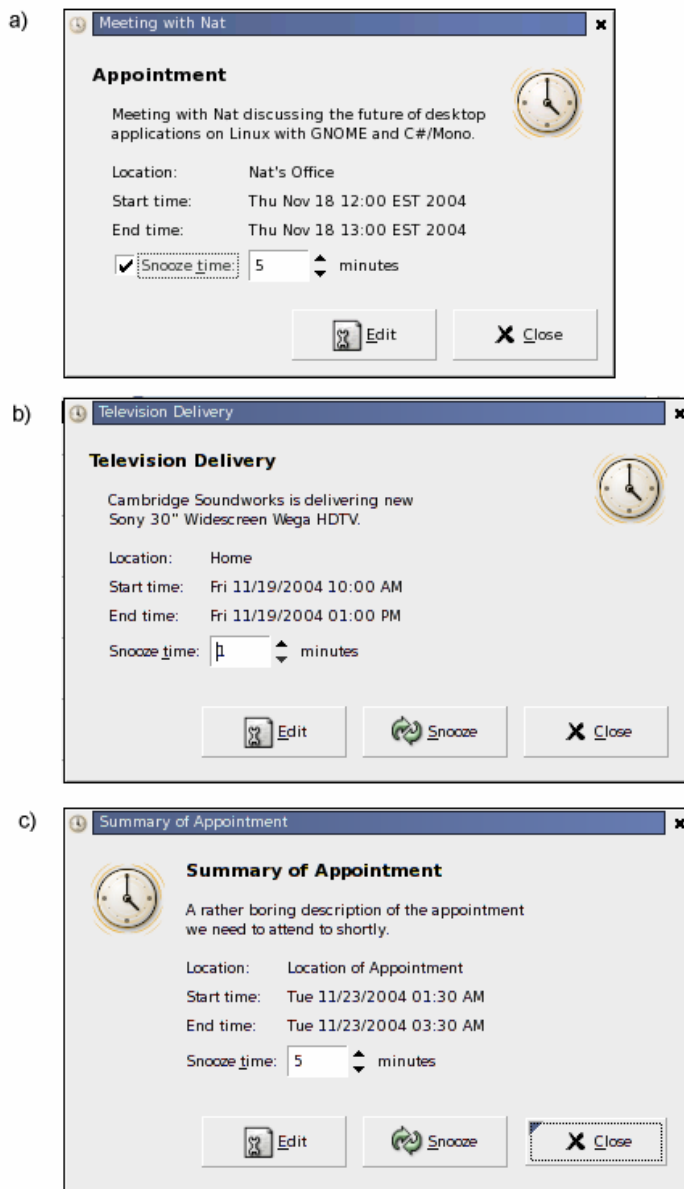


Figure 6. Incremental re-design of the *Evolution* alarm dialog - from the *Evolution* Blog (<http://codeblogs.ximian.com/blogs/evolution>)

The design in Figure 6a was posted to the *Evolution* blog on 18th November 2004 with the following text:

New Alarm Dialog

Worked on creating a mock-up for a re-design of the alarm notification dialog in *Evolution* today, and after a few revisions, Ben and I agreed on the

image below. I'm working on the code to get this implemented and into CVS, so it should be popping up on your desktop soon.

At the time of writing this blog entry has 189 attached comments – however most of these are 'comment spam' and do not relate to the proposed design. We estimate that 16 comments are 'real' and relate to the *Evolution* application. Sample quotes from these comments include: 'In Gnome, icons usually go on the left side of alert windows', 'what happens when I have more than one reminder? Will I have multiple dialogs ...?' and 'I still feel that the basic Gnome alarm+calendar functionality shouldn't depend on a whole email app'. Thus, comments range from low-level widget placement through possible scenarios to inter-application dependencies. Also, one comment described a re-design from a HCI class who had taken on the alarm dialog as a discussion topic.

The next day this text accompanied the posting of Figure 6b:

New Alarm Dialog Again

After my previous blog entry about the Alarm dialog mock-up, a lot of feedback was had, and I went off to implement a version. What I currently have implemented is in the screenshot below. If anyone knows a standard localizable way to put the time/date information into a single line, please, e-mail me. This version has the times as currently displayed by Evolution, as well as the appointment summary in bold, instead of just an "Appointment" label, and single-click snoozing

This entry has 7 real comments including comments such as 'I don't seem to understand when the alarm is set to go off', 'thank god someone is finally doing something about the alarm dialog' and 'I do agree with the comment above though that said the title and time should be larger.'

Finally, this text accompanied the design shown in Figure 6c on November 23rd 2004:

New Alarm Dialog in CVS

Finally, the new alarm dialog is in CVS. A screenshot of what was committed is below. The icon was moved to the left side, and the secondary text/widgets were re-aligned to match the HIG [Human Interface Guideline] 2.0 Alert specs more precisely. This cleans up the code quite a bit and gets rid of the HTML widget that we were using before.

This entry has 6 real comments including: 'I hope (expect) the date/time convention follows the locale settings' and a confirmation that it does.

Previous work (e.g. Herbsleb and Mockus 2003) has identified the variety of tools used by globally distributed teams to support work activities; we believe this is the first reported observation, in the literature, of design occurring via a blog.

Analysing Design-by-Blog

There are several interesting elements about this 'design by blog' episode, which seems to address several of the problems outlined earlier, and which we believe generalise:

- It is an example of a small design team (in this case presumably 1 person) presenting ideas and reacting to feedback, allowing for the exploration of a solution while maintaining some design consistency. It is not really design-by-committee. This, and all the examples of design by blog we know about are actually re-design – something is already, it does not start with formal requirements, scenarios or anything similar – it starts with code (Raymond 1998).
- It happens quickly; over a few days. This is in almost all ways ideal, but it can mean that certain people with valuable insights may not see it in time to contribute
- Design is more public and accessible than Bugzilla. Commenting does not require the rather complex registration process of Bugzilla. Unfortunately, public discussions can be targeted with comment spam – this is a risk of doing things in public, and a side effect of the ease of participating compared to Bugzilla. Blog use is also more public and transparent than taking a Bugzilla design discussion offline to a face-to-face meeting or private electronic communication. The public interface inspection nature of interface design-by-blog can be regarded as a HCI variant of code inspection (Lanubile *et al.* 2003, Lussier 2004).
- The graphics are inline in the blog postings, so are automatically the main object for discussion. However, the incorporation of graphics is asymmetric. Comments are all textual (apart from the HCI class re-design who used a separate web server). Finding the right place for the user's area of interest within the blog is also easier, but only because design blogs are far smaller, containing far fewer than the thousands of entries in Bugzilla that necessitate a highly complex search interface.
- Blogs as a genre have a conversational style amenable to the more provisional, discursive, argumentative, iterative nature of interface design. This can be contrasted with the complexity-management focus on workflow processing in Bugzilla that enables very large numbers of bugs to be reported, analysed, fixes proposed, and solutions reviewed and approved. The *Evolution* blog follows the typical blog genre of many different items organized temporally. Different designs are explored at different times in the blog, along with information about bug fixes, updates, news, help requests, code questions, etc. It is a far more informal organization than the indexing of Bugzilla. We assume (for now) that people used to the blog genre do not have major difficulties with such heterogeneous collections.

Blogs create a space to show the evolution of design ideas in a way that does not clog up the already complex bug tracking and workflow procedures of systems such as Bugzilla. This additional space allows an interleaving of design ideas, rationales and critiques. As well as being very interesting to track how a design evolved for researchers or students of HCI, it can also help in the immediate design process. Documenting design rationales is acknowledged to be important, but difficult to persuade busy designers to undertake. Knowing that there is a current audience reading one's rationale rather than a hypothetical future code maintainer can be a great motivator, particularly when this audience responds with comments. Additionally, alternate designs, their rationales and

the reasons for their ultimate rejection, can be a valuable element in achieving consensus about the final design (Berkun 2000).

The various advantages and disadvantages of the current manifestations of design-by-blog can be useful in informing both the development and use of alternate technologies and infrastructures and in developing technological and organizational fixes to current problems. For example, it would not be difficult for an OSS project to develop a blog-harvester to accumulate, index and archive multiple blogs used in the design process.

Finally, we note that the distributed micro-level design activity of blog use contrasts with the centralized workflow management system of Bugzilla in interesting ways. Both aim to address the problem of complexity management and each have their own strengths and weaknesses as we have shown. It is perhaps ironic, and certainly contentious, to note the parallels and distinctions between a centralized (perhaps Cathedral-like?) structure such as Bugzilla and a distributed (perhaps Bazaar-like?) structure such as many different design blogs (Raymond 1998).

5. DISCUSSION

We divide the discussion into two parts: how OSS usability compares with other aspects of OSS development and secondly, how OSS usability compares with industry best practice.

5.1 Comparing OSS usability to other OSS processes

Open source development relies on a variety of tools: compilers, mailing lists, build systems, version control systems, memory leak detectors, bug repositories etc. Many of these tools automate tasks where manual alternatives would be infeasible. These tools are used by globally distributed developers to coordinate their work and communicate their results. We have shown how usability activities relate to elements of the open source infrastructure and are moving to populate others such as blogs. In the light of this extensive tool use it is perhaps surprising that open source projects appear not to be using any tools, beyond interface construction tools such as *Glade*, to aid their usability evaluation. However, the HCI community in general has not developed any significant automated approaches for non-Web applications.

Hackers will generally (and sometimes religiously) only use OSS tools to develop OSS code, but few such tools exist to help developers design guideline-compliant applications.

Benson (2004)

We consider that research into usability-oriented evaluation tools could be extremely valuable for OSS communities that lack human usability expertise (Nichols and Twidale 2003).

The 'many eyeballs' approach to open source code improvement (Raymond 1998) relies on the right eyeballs observing the code and then taking some corrective action. As such it has similarities with the concept of globally-distributed software inspection (Lanubile *et al.* 2003). Interfaces are slightly different as users experience them directly

but many are not in the position to act on their observations. Although the number of developers who have sufficient knowledge to 'scratch their itch' maybe small, potentially *all* users can contribute to an OSS community through usability reporting. The remaining 90% of passive members' (Nakakoji *et al.* 2002) represent a huge untapped resource for OSS; one that can be engaged by the right infrastructure. Developments such as design-by-blog and *Hendrix* show moves are being made in this direction.

5.2 Comparing OSS usability to industry 'best practice' usability methods

In terms of usability, there is a clear divide between the small OSS projects that epitomise Raymond's (1998) 'developers scratching an itch' and the large projects that have company support (Benson *et al.* 2004). The large supported projects are adopting techniques from mainstream HCI and appear to be benefiting. The GNOME HIG are now a key infrastructural resource that guides development and streamlines discussions. A problem with the interface can be legitimated by referral to the HIG. Any disagreement can really only be about whether it does not in fact deviate from the HIG. By contrast, in projects lacking clearly specified guidelines, a problem has to be noted as deviated from the reporter's notion of usability norms, and both the problem and the norm are subject to debate. The consequence seems to be the rather unstructured discussions that are noted in section 3.

Most Human Computer Interaction techniques assume co-located work both in usability analysis and interface design, whether it is formal usability tests, ethnography, think-aloud protocols, heuristic analyses, cognitive walkthroughs, paper prototyping, participatory design, or the use of toolkits. There is growing interest in distributed software development in general (e.g. Lanubile *et al.* 2003, Prikladnicki *et al.* 2003). Similarly there is a growing interest in remote usability testing as a way to address the problems of assembling sufficient numbers of subjects, and creating ecologically valid evaluations (Wichansky 2000). Far less attention has to date been given to distributed usability design and work in this area could greatly benefit OSS.

6. CONCLUSION

We have attempted to show how usability work in OSS projects needs different processes than regular work on functionality, fixing crash bugs and improving reliability. From a series of small scale studies of a number of OSS projects having a significant concern for end user usability, we have identified a set of issues that lead to informal requirements for a usability infrastructure. Although no such infrastructure exists with the sophistication of the functionality bug workflow tracking of for example Bugzilla, we believe that much can be learned from the rather ad hoc appropriations of available technologies by OSS usability analysts and interface designers. In particular we have explored how the use of design-by-blog and human interface guidelines offer particular advantages in widening participation and yet maintaining focus. This is critical in a part of software development where subjectivity and debate are inevitable and valuable but which in excess can prevent efficient progress in improving the overall user experience.

REFERENCES

Austin RD, 2001. The Effects of Time Pressure on Quality in Software Development: An Agency Model, *Information Systems Research* 12(2) 195-207.

Benson C. 2004. Meeting the challenge of open source usability. *Interfaces*. Autumn 2004. 9-12.

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

Berkun S. 2000. Why Good Design Comes from Bad Design
<http://www.uiweb.com/issues/issue08.htm> <http://webtools.mozilla.org/hendrix/> accessed 28 February 2005.

Feller J, Fitzgerald B. 2002. *Understanding Open Source Software Development*. London: Addison-Wesley.

Frishberg N, Dirks AM, Benson C, Nickell S, Smith S. 2002. 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. 932-933.

German DM. 2003. The GNOME Project: a Case Study of Open Source, Global Software Development, *Software Process Improvement Practice* 8(4) 201-205.

Gilb, T. 1989 Deadline pressure: how to cope with short deadlines, low budgets and insufficient staffing levels, in Boehm B. (ed.) *Software Risk Management*. Piscataway, NJ, USA: IEEE Press.

Hammontree M, Weiler P, Nayak N. 1994. Remote Usability Testing. *interactions* 1(3) 21-25.

Hars A, Ou S. 2002. Working for Free? Motivations for Participating in Open-Source Software Projects, *International Journal of Electronic Commerce*, 6(3) 25-39.

Hendrix, 2004 Mozilla.org <http://webtools.mozilla.org/hendrix/> accessed 28 February 2005.

Herbsleb JD, Mockus A. 2003. An Empirical Study of Speed and Communication in Globally Distributed Software Development, *IEEE Transactions on Software Engineering*, 29(6) 481-494.

Hertel G, Niedner S, Herrmann S. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel, *Research Policy* 32(7) 1159-1177.

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

Nakakoji K, Yamamoto Y, Nishinaka Y, Kishida K, Ye Y. 2002. Evolution Patterns of Open-Source Software Systems and Communities, *Proceedings of the Workshop on*

- Principles of Software Evolution, International Conference on Software Engineering*. New York: ACM Press, 76-85.
- Lanubile F, Mallardo, T, Calefato F. 2003. Tool Support for Geographically Dispersed Inspection Teams, *Software Process Improvement Practice* 8(4) 217-231.
- Lussier, F. 2004. New Tricks: How Open Source Changed the Way My Team Works, *IEEE Software* 21(1) 68-72.
- Mockus A, Fielding RT, Herbsleb J, 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 11(3) 309-346.
- Nardi BA, Schiano DJ, Gumbrecht M, Swartz L. 2004. Why we blog. *Communications of the ACM* 47(12) 41-46.
- Nichols, DM, McKay D, Twidale MB. 2003 Participatory Usability: supporting proactive users, *Proceedings of 4th ACM SIGCHI NZ Symposium on Computer-Human Interaction (CHINZ'03)*, Dunedin, New Zealand: SIGCHI New Zealand. 63-68.
- Nichols DM, Twidale MB. 2003 The Usability of Open Source Software, *First Monday* 8(1) 2003. http://firstmonday.org/issues/issue8_1/nichols/, accessed 28 February 2005.
- Nielsen, J. 1993. *Usability Engineering*, Boston, MA, USA: Academic Press.
- Prikladnicki R, Audy JLN, Evaristo R. 2003. Global software development in practice lessons learned, *Software Process: Improvement and Practice* 8(4) 267-281.
- O'Mahony S, 2003. Guarding the commons: how community managed software projects protect their work, *Research Policy* 32(7) 1179-1198.
- Raymond ES, 1998. The Cathedral and the Bazaar, *First Monday*, 3(3) (March), at http://firstmonday.org/issues/issue3_3/raymond/, accessed 28 February 2005.
- Raymond ES. 1999. "The revenge of the hackers," In: M. Stone, S. Ockman, and C. DiBona (editors). *Open Sources: Voices from the Open Source Revolution*. Sebastopol, Calif.: O'Reilly & Associates, pp. 207-219.
- Sandusky RJ, Gasser, L, Ripoche G. 2004. 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. 80-84.
- Scacchi, W. 2002. Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings — Software* 148(1) 24-39.
- Scacchi, W. 2004. Free and Open Source Development Practices in the Game Community, *IEEE Software*, 21(1) 59-66.
- Smith S, Engen D, Mankoski A, Frishberg N, Pedersen N, Benson C. 2001. *GNOME Usability Study Report*, Sun GNOME Human Computer Interaction (HCI), Sun Microsystems, Inc., at http://developer.gnome.org/projects/gup/ut1_report/report_main.html, accessed 28 February 2005.

Trudelle P. 2002. 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., at http://www.iol.ie/~calum/chi2002/peter_trudelle.txt, accessed 28 February 2005.

Twidale MB, Nichols, DM. 2005. Exploring usability discussions in open source software, *Proceedings of the Thirty-Eighth Annual Hawaii International Conference on System Sciences (HICSS'05)* (CD-ROM). IEEE Computer Society Press. Eight pages.

von Hippel E, von Krogh G. 2003. Open source software and the "private-collective" innovation model, *Organization Science* 14(2) 209-223.

Weber, S. 2004. *The Success of Open Source*. Cambridge, MA, USA. Harvard University Press.

Wichansky AM. 2000. Usability testing in 2000 and beyond. *Ergonomics* 43(7) 998-1006.

Yamauchi Y, Yokozawa M, Shinohara T, Ishida, T. 2000. Collaboration with Lean Media: how open-source software succeeds, *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'00)*, New York, NY: ACM Press. 329-338.