Participatory Usability: supporting proactive users

David M. Nichols, Dana McKay
Department of Computer Science
University of Waikato
Hamilton, New Zealand
{dmn, dana}@cs.waikato.ac.nz

Michael B. Twidale
Graduate School of Library
and Information Science
University of Illinois at Urbana-Champaign
IL 61820, USA
twidale@alexia.lis.uiuc.edu

ABSTRACT
After software has been released the opportunities for users to influence development can often be limited. In this paper we review the research on post-deployment usability and make explicit its connections to open source software development. We describe issues involved in the design of end-user reporting tools with reference to the Safari web browser and a digital library prototype.

Keywords
Usability, open source, critical incident reporting

INTRODUCTION
Usability has a firm place within human-computer interaction (HCI) and an increasingly prominent role in software development. Many methods have been developed for improving the usability of software and devices. Usability techniques range from methods for improving requirements gathering (such as ethnography) to automated analysis of interface code. Most of these methods fall under the term 'user-centred design' and are intended to improve products before they are deployed.

The role of the user in user-centred design is generally as a design participant, a usability study participant or the object of field study by ethnographers. Once the software is deployed and actually in use then the opportunities for remote users to influence future development are rare.

In this paper we review the research on post-deployment usability and make explicit its connections to open source software development. We describe the requirements of end-user reporting tools with reference to the Safari web browser and a digital library prototype. This prototype serves as an interesting test case for the challenge of designing the interface to a novel piece of functionality in order to make both its use and purpose clear to prospective users. We outline how social considerations, in the form of privacy concerns, need to be represented at the interface of post-deployment usability tools.

USABILITY WITHIN HCI
HCI as a discipline has under-emphasized the importance of post-deployment usage usability activities. For laudable reasons, a major thrust of HCI has been to attempt to influence the early stages of design, right from the requirements capture stage. It is known from software engineering that the earlier a bug can be caught in the design process, the cheaper it is to fix it. The cost savings can amount to orders of magnitude. The same is also true of usability bugs, both problems with users learning and using the application, and failures of the application to mesh with actual user needs and workplace practice.

As a result, HCI researchers and advocates have focused on the need for early involvement in the specification and prototyping processes, rather than leaving the interface to the final stages just prior to deployment and when the design is already committed to particular functionality decisions.

There are a few pieces of work that deal with post-deployment usability but the overwhelming majority of HCI work concerns itself with pre-deployment methods. This was perhaps understandable when it was more difficult and costly to communicate with users in the 'real world', thus making pre-deployment the only opportunity for usability engineering. However, changes in computing now allow for much closer post-deployment user contact.

In particular, there are three trends in software use that combine to provide an opportunity for post-deployment methods to be an increasingly effective addition to usability methods:

- Low-cost reliable networking connecting users and developers
- Incremental software versions—that is, the software application interface most users work with is a relatively small change over a widely-used previous version
- Easy upgrades for end users, either via simple downloads, or by virtue of the application itself being a remote access to a central software resource, such as a digital library with a web-based interface

POST-DEPLOYMENT USABILITY
Typically, once a piece of software has been released users have reduced opportunities to communicate with developers. The rich, time-consuming interactions of the user-centred design process are largely over. The problems that users experience are often resolved via an ad-hoc combination of online help materials, local technical support, software support telephone help lines and informal interactions with other users. However there are many

frustrating episodes that never make it to any of these help resources—individual users just get frustrated and develop personal work-around tactics.

From the point of view of the developers these incidents are invisible; this is one reason why alternative strategies (such as ethnography) for uncovering this type of incident have proven so valuable. Observational studies *in situ* have been the only way for developers to get a glimpse of the everyday life of typical users struggling with their software.

User Frustration

Recent studies of user frustration [2, 14] have highlighted the everyday annoyances of computer-based work. A perhaps surprising result was that subjects estimated "one-third to one-half of the time spent in front of the computer was lost, due to frustrating experiences" [14].

The question we wish to pose is: what should the computer program do at the 'moment of frustration'? Although long term generic responses such as 'improve the design' are reasonable we suggest 'moments of frustration' are likely to continue to occur as users' expectations and behaviour changes. Possible immediate software responses include:

1. Do nothing; most software in use today is simply passive. Some applications may refer the user to a help system, associated web resources or offline help—if they are available.
2. Support users to manage and recover from negative emotional states [13]. This is part of the relatively new field of affective computing [18].
3. Enable the user to dynamically change the program to attempt to resolve the problem by themselves [5].
4. Suggest the user improve the source code of the program. This is the open source solution.
5. Communicate details of the frustrating incident back to the developers. This is post-deployment usability, and it includes both end-user reporting and instrumented software.

These different responses reflect different philosophies about interaction. The standard response (no. 1) is saying that you have a piece of software, use it as it is or wait for the next version. The affective response (no. 2) says that the software can go further and react to the state of the user. Recent research has considered explicitly designing computers to respond to user frustration and found that users do respond positively to affective support [13].

We hypothesize that part of the frustration users experience may be alleviated by providing an outlet for their complaints (response no. 5). Although Klein *et al.* [13] found that their 'vent' condition (letting users express their frustration) was less effective than their 'affective support' condition we believe this is because the venting was not directed at product improvement. Most users have no such frustration outlet and no easy means to complain directly back to the developers.

Response 3 suggests an engineering paradigm—that we can radically alter the way users interact with software by creating an application that is significantly more flexible at run-time than current software. This approach also envisages users taking on more power and responsibility.

The final two responses also allow users to have a more active role in software development. The open source solution is to encourage users to fix the problem themselves—this is the motivation of "scratching a personal itch" [19]. The vision is of developer-users interacting with the source code to quickly achieve a new software version which can be shared with other users. The restriction here is that users need to be very technically competent and so this option is only realistic for a small proportion of the user base [17]. Although open source programs don't actually prompt users to contribute source code, this message ('fix it yourself') is firmly embedded in the culture.

The final response (no. 5) is to acknowledge that most users are not skilled enough to adopt the open source solution but are still knowledgeable about their own interactions. This is a key pre-requisite of the user reporting element of post-deployment usability [4, 7].

Approaches to Post-Deployment Usability

Post-deployment usability is a subset of remote usability [6, 8] in that it requires many remote users to participate as part of their *normal* usage of computer software.

The actions of users can be automatically recorded and sent back to developers through instrumented software. Such remote usage recording is now widespread in many devices including vending machines and elevators. Alternatively, the software can allow the user to explicitly send a report of a usability incident [4, 7]. In this paper we concentrate on this second option; user-initiated reporting.

The research on frustration suggests that there is no shortage of raw material, but in general there is no infrastructure for users to relay incidents back to developers. The reader may like to try to locate an avenue for complaint in their software—we predict it will take some time to locate such a facility (if it exists), that it will probably require another piece of software to use (e.g. email or a web browser) and will require the user to enter information already available to the computer.

The difficulty of complaining via software is illustrative of the developer's attitude: individual user frustrations are deemed not valuable enough to attempt to record. Yet the same developers may very well promote the usability laboratory testing of their products as a selling point—which in many cases is simply a more expensive method of getting at the same underlying issues.

Klein *et al.* note that for many corporations the idea of admitting a product's failure may be counter-intuitive—and even legally unwise [13]. Open source software (see below) uses a completely different world view. Our view is that explicitly designing for these failure situations and allowing users a positive outlet can capture some of the incident data that is otherwise lost. Additionally, it may even help users to feel a little better. In other enterprises, particularly services and the hospitality industry, complaints and customer feedback are actively encouraged and part of the culture [11]. Indeed Shneiderman [21] and Mayhew [15] both suggest that users could be given financial incentives for contributing usability reports.

The value of collecting usability data from real users as part of their day-to-day activities has been clearly recognized [4, 7]. Hartson and Castillo note the techniques' cost effectiveness and the lack of the need for an evaluator [7]; this both simplifies the data collection and reduces the potential biases of evaluator-effects on the users [1]. The major piece of research on end-user reporting indicates that users are fairly successful in identifying critical incidents and judge their severity in roughly the same way as experts [7].

Reports in post-deployment usability consist of two main elements: objective program state information [9] and subjective comments provided by users. Reporting may be initiated by the user [7] or prompted by the application [10]. To date there have no reports in the HCI literature of widespread trials of usability-oriented user reporting. However open source projects have successfully involved many distributed users in contributing to bug databases.

Open Source Usability

The usability of many open source products is widely regarded to be worse than their commercial counterparts. There are many possible reasons for this including resources, processes, the nature of voluntary contributions, development team composition etc. [17]. However the key element, for this paper, is that open source projects do manage to engage their users in the development process. User participation comes in several forms: code such as patches, documentation, testing and bug reporting [17].

Users are allowed to be proactive in open sources, instead of being 'couch potatoes' [5] they become an active part of the software development. For most users the majority of work on an open source project is too technically complicated—but bug reporting is different. A system such as Bugzilla [3], the bug database of the Mozilla Project [16], holds over 200,000 bug reports from users all over the world. Although Bugzilla users are technically adept it is not a requirement for bug reporting—however the complexity of the Bugzilla reporting tool itself is a barrier for many potential users.

Open source projects have shown that distributed networked users can contribute to software development. In the rest of the paper we examine how we may be able to adapt that participatory ethos from functionality to usability.

USER REPORTING OF BUGS AND CRASHES

Although usability-centred end-user reporting is not present in everyday applications several bug reporting tools have made their way onto the desktops of everyday users.

Crash Reporting

Netscape, Mozilla and Microsoft Windows XP all contain crash reporting tools that send information back to the developers. These tools are invoked when a program crashes and typically have little user input. As the program is no longer running when the data (such as which threads were running) is gathered, any information has limited usability value. Indeed, the current use of such tools focuses on functionality errors rather than usability errors. The difference is that the latter are rather more subjective. A crash is a crash, but a cause for confusion for one user depends on the goal of the user using the software, the context of the task and the user's prior experience. However, these tools do demonstrate the technical feasibility of widespread networked feedback based on users' interactions.

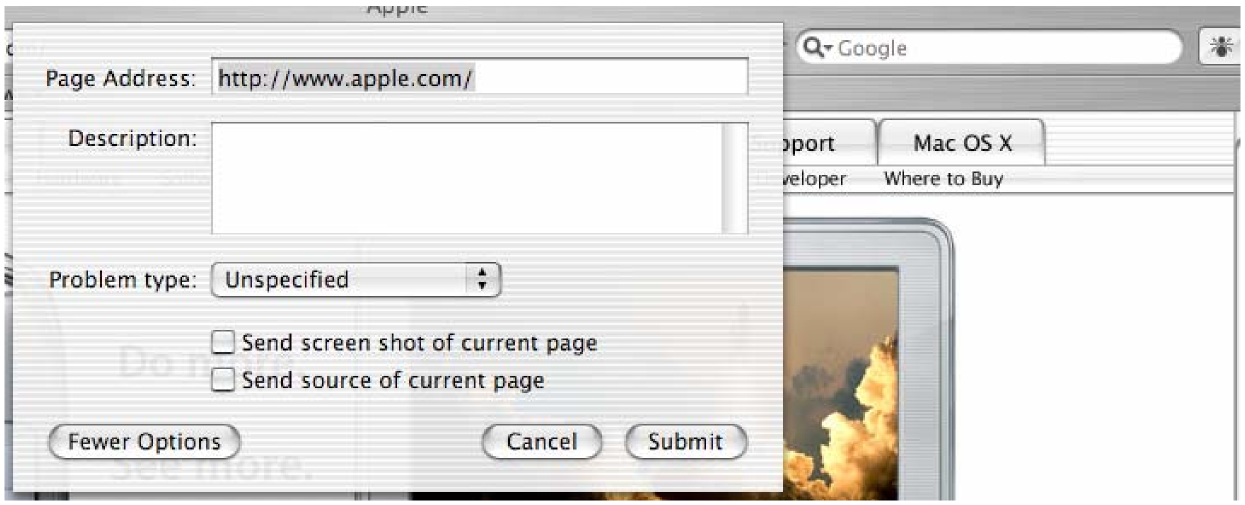
Safari

Safari [20] is a recently released web browser, still in public beta testing (a well established quasi post-release evaluation, again chiefly of errors and functionality), developed by Apple Computer, Inc. for Mac OS X. It is based around the open source KHTML rendering element used in the KDE Project [12]. Apple has included a user reporting element in Safari to help the developers identify web pages that do not display correctly.

A user initiates a report in Safari by clicking the 'bug' symbol—shown on the top right of Figure 1. The dialog shown on the left of Figure 1 is then displayed. In the next section we use Safari as an example in discussing the design of end-user reporting tools.

DESIGN OF REPORTING TOOLS

The previous sections have outlined the landscape of reporting tools for post-deployment usability. In this section we describe the key design challenges in implementing such tools with reference to Safari, Bugzilla and the crash reporting tools:



Page Address: http://www.apple.com/

Description:

Problem type: Unspecified

☐ Send screen shot of current page

☐ Send source of current page

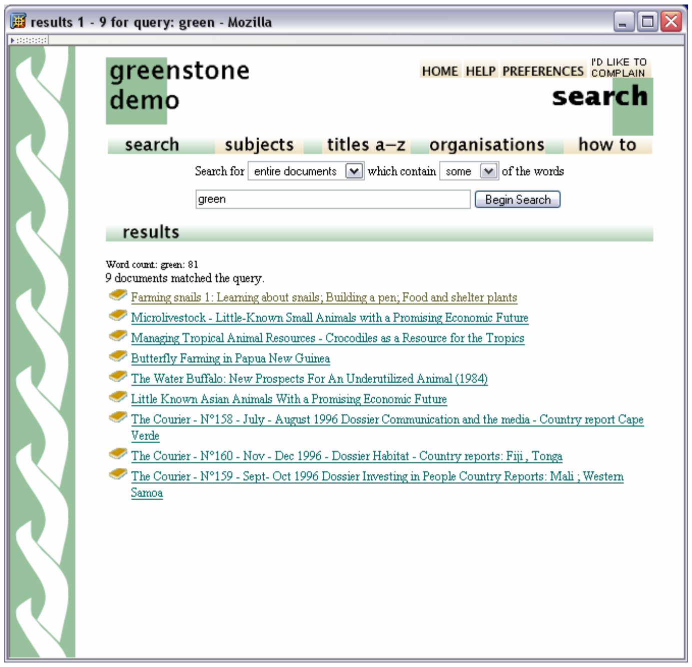
Figure 1. User Reporting in Safari. Safari is a web browser developed by Apple Computer, Inc.

- *Ubiquity*. As Shneiderman suggests, facilities for end-user reporting should be ubiquitous as we cannot predict when problems may arise [21]. In Safari the 'bug' is always visible in the main browser window and, as the facility is restricted to commenting on HTML rendering, it doesn't need to be visible in, say, the preferences dialog window. A web-based system is always available but requires extra work to invoke.
- *Registration & Anonymity*. The Safari 'bug' does not require any form of registration and in this respect functions much like the crash reporting tools mentioned earlier. This feature of 'anonymous reporting' lowers the cost (both in terms of effort and privacy) of reporting when compared with a public registration-based system like Bugzilla.
- *Application Integration*. The Safari tool is integrated into the main application. This integration allows it to offer an option of recording screen shots—an option not available to the crash tools or a separate web-based system like Bugzilla. Hartson and Castillo report that their participants preferred the reporting tool to be integrated with the application [7]. The web-based reporting systems also force users to input information that is available to the computer; this raises the cost of reporting and contributes to inaccuracy in the reports.
- *Ease of Use*. The Safari tool, like the crash reporters, submits the report with just one click, all other information is optional. This contrasts with an average of five minutes to fill in a full critical incident report [7]. We believe that two clicks, one to start the tool and one to send a report, are all the user effort that should be necessary for a minimal report.
- *Subjective & Objective Information*. Safari combines low-cost objective information (e.g. the URL) and subjective information (user comments) into one report. Reporting tools need to provide both types for developers to efficiently interpret the report.
- *Tracking a Report*. Of the systems discussed only the open source Bugzilla allows a report to be tracked after it has been sent. Hartson and Castillo [7] note that the participants in their study wanted feedback on their reports and Bugzilla allows anyone to track the progress of a report.

PROTOTYPE

We have a developed a prototype reporting tool for the Greenstone digital library software [22]. Our aim here is to show how the design challenge described above was addressed in the current version of the prototype (which itself will of course evolve in the light of user feedback). Greenstone differs from Safari in that it presents its interface in via a web browser; whereas Safari is a native application with a full range of interface options available. Consequently, the reporting mechanism is constrained to use web technologies—and so is more restricted than the integrated tool in Safari.

Figure 2a shows a typical part of digital library collection presented via Greenstone. To initiate a report the user clicks the "I'd like to complain" button at the top right of Figure 2a. The report button element is present on all the pages within a collection allowing incidents to be generated from any point in the system. The incident report interface is shown in Figure 2b. This interface is designed to allow the user to contribute with minimum effort on their part—while still providing Greenstone developers with enough information to assess the problem, and propose a solution. Users can provide optional subjective feedback but this


results 1 - 9 for query: green - Mozilla

greenstone demo

HOME HELP PREFERENCES I'D LIKE TO COMPLAIN

search

search subjects titles a–z organisations how to

Search for entire documents which contain some of the words

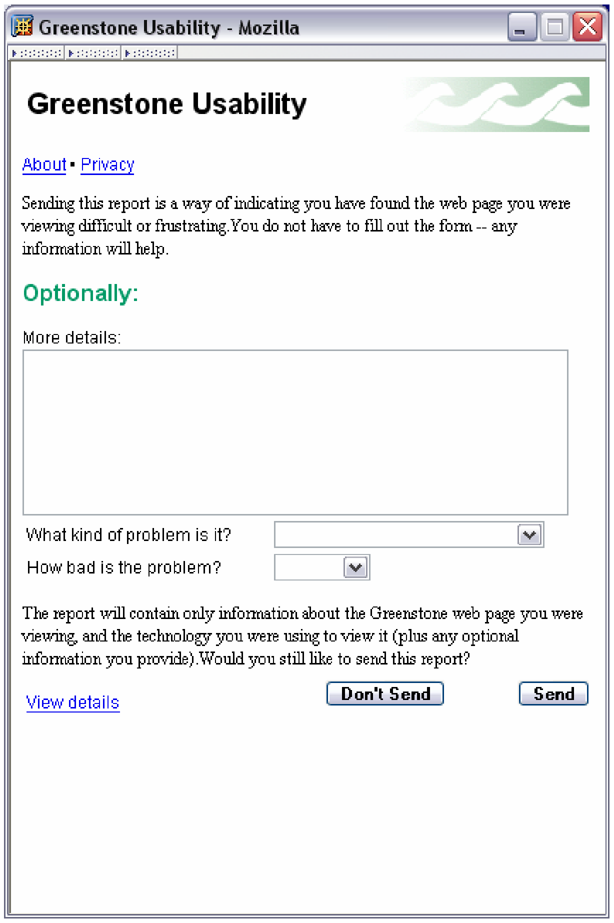
green Begin Search

results

Word count: green: 81
9 documents matched the query.

- Farming snails 1: Learning about snails; Building a pen; Food and shelter plants
- Microlivestock - Little-Known Small Animals with a Promising Economic Future
- Managing Tropical Animal Resources - Crocodiles as a Resource for the Tropics
- Butterfly Farming in Papua New Guinea
- The Water Buffalo: New Prospects For An Underutilized Animal (1984)
- Little Known Asian Animals With a Promising Economic Future
- The Courier - N°158 - July - August 1996 Dossier Communication and the media - Country report Cape Verde
- The Courier - N°160 - Nov - Dec 1996 - Dossier Habitat - Country reports: Fiji , Tonga
- The Courier - N°159 - Sept- Oct 1996 Dossier Investing in People Country Reports: Mali ; Western Samoa

(a)


Greenstone Usability - Mozilla

Greenstone Usability

About • Privacy

Sending this report is a way of indicating you have found the web page you were viewing difficult or frustrating. You do not have to fill out the form -- any information will help.

Optionally:

More details:

What kind of problem is it?

How bad is the problem?

The report will contain only information about the Greenstone web page you were viewing, and the technology you were using to view it (plus any optional information you provide). Would you still like to send this report?

View details Don't Send Send

(b)

Figure 2. User Reporting Prototype in Greenstone.

interface doesn't reveal the key elements of report construction.

The majority of the information gathered by the reporting tool is via JavaScript—information about the user's browser, operating system, screen resolution and the time they took using the reporting interface. The state of form elements on the Greenstone interface is also captured; including any query terms in the search text-field.

The prototype tool is ubiquitous within a Greenstone collection, allows report submission with minimal user effort (two mouse clicks) and supplements user comments with objective program state information. As previous research using laboratory-based studies [7] hasn't tested reporting as part of users' everyday activities, we intend to deploy this tool into Greenstone and observe how users respond to a reporting infrastructure

PRIVACY

Whilst a reporting interface can appear to be relatively innocuous the program is clearly collecting information relating to the user's interactions and sending it somewhere on the network.

The user has to trust that the program is not sending any sensitive information, and only the user can determine what is sensitive to them. Clearly no user is going to be pleased with their credit card details being collected, but whether they care about the page they are viewing being noted will most likely depend on the content. Most users will not mind a page about the weather being logged, but they may be more concerned about a page containing, for example, sensitive medical information.

This need for privacy is sometimes in conflict with developers getting the information they need to solve problems; in the Safari example the content of the page is necessary to determine what doesn't render properly. It could be argued that the user relinquishes their privacy when the software is installed—on a purely technical level this is true, software is capable of recording user information from that point on. A typical user has no way of proving that a piece of software is not transmitting the contents of their filestore across the Internet.

The key issue in the conflict between developer needs and user privacy is trust—the user has to trust the developer not to take nor disseminate potentially sensitive information.

To generate trust the developer must allow the user to see what information is being recorded, and provide an explicit policy on how this information will be used. This is the place of the "Privacy" and "View Details" links in Figure 2b. Without this transparency of information and intent, reporting mechanisms are unlikely to gain the social acceptance necessary for their widespread adoption.

CONCLUSION

In this paper we have reviewed the, relatively small, literature on end-user reporting and linked it to both open source development and current prototype systems.

We believe that empowering end-users to proactively contribute to usability activities is a valuable, and under-explored, technique. Open source communities have shown that distributed development can achieve rapid results in terms of functionality; a challenge for HCI is to explore whether we can achieve similar success via participatory usability.

ACKNOWLEDGMENTS

We thank Ben Shneiderman, Jenny Preece, Matt Jones, George Buchanan, Stuart Yeates and Rob Akscyn for their valuable comments on the topics in this paper.

REFERENCES

1. Bentley, T. Biasing Web Site User Evaluations: a study. *Proceedings of the Conference of the Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia (OzCHI2000)*. (2000) Sydney, Australia, 130-134.
2. Bessiere, K., Ceaparu, I., Lazar, J., Robinson, J., and Shneiderman, B. *Understanding Computer User Frustration: Measuring and Modeling the Disruption from Poor Designs*, Technical Report HCIL-2002-18 (2002) Human–Computer Interaction Lab, University of Maryland, USA.
3. Bugzilla. at http://bugzilla.mozilla.org [accessed 30th March 2003]
4. Castillo, J.C. Hartson, H.R. and Hix, D. Remote Usability Evaluation: Can Users Report Their Own Critical Incidents? *Proceedings of the Conference on Human Factors on Computing Systems (CHI'98): Summary.* (1998) ACM Press, 253-254.
5. Fischer, G. Beyond "Couch Potatoes": From Consumers to Designers and Active Contributors. *First Monday* 7,12 (December 2002) at http://firstmonday.org/issues/issue7_12/fischer/
6. Hammontree, M., Weiler, P. and Nayak, N. Remote Usability Testing. *interactions* 1, 3 (1994) 21-25.
7. Hartson, H.R. and Castillo, J.C. Remote Evaluation for Post-Deployment Usability Improvement. *Proceedings of the Conference on Advanced Visual Interfaces (AVI'98)* (1998) ACM Press, 22-29.
8. Hartson, H.R., Castillo, J.C., Kelso, J and. Neale, W.C. Remote Evaluation: The Network as an Extension of the Usability Laboratory. *Proceedings of the Conference on Human Factors in Computing Systems (CHI'96)* (1996) ACM Press, 228-235.
9. Hilbert, D.M. and Redmiles, D.F. Extracting Usability Information from User Interface Events. *ACM Computing Surveys* 32, 4 (2000) 384-421.
10. Hilbert, D.M. and Redmiles, D.F. Large-Scale Collection of Usage Data to Inform Design. *Human-Computer Interaction — INTERACT'01: Proceedings of the Eighth IFIP Conference on Human-Computer Interaction (2001)* Tokyo, Japan, 569-576.
11. Johnston, R. and Mehra, S. Best-practice complaint management. *Academy of Management Executive* 16, 4 (2002) 145-154.
12. KDE Project at http://www.kde.org [accessed 30th March 2003]
13. Klein, J., Moon, Y. and Picard, R.W. This computer responds to user frustration: theory, design and results. *Interacting with Computers* 14 (2002) 119-140.
14. Lazar, J., Bessiere, K., Ceaparu, I., Robinson, J. and Shneiderman, B. Help! I'm lost: user frustration in web navigation. *IT & Society* 1, 3 (2003) 18-26.
15. Mayhew, D.J. *The Usability Engineering Lifecycle: a practitioner's handbook for user interface design*. Morgan Kaufmann, San Francisco, CA, 1999.
16. Mozilla Project. at http://www.mozilla.org [accessed 30th March 2003]
17. Nichols, D.M. and Twidale, M.B. The usability of open source software. *First Monday* 8, 1 (January 2003) at http://firstmonday.org/issues/issue8_1/nichols/
18. Picard, R.W. *Affective Computing*. MIT Press, Cambridge MA, 1997.
19. Raymond, E.S. The Cathedral and the Bazaar. *First Monday*, 3, 3 (March 1998), at http://firstmonday.org/issues/issue3_3/raymond/
20. Safari, Apple Inc. at http://www.apple.com/safari [accessed 30th March 2003]
21. Shneiderman, B. *Leonardo's Laptop: Human Needs and New Computing Technologies.* MIT Press, Cambridge MA, 2002.
22. Witten, I.H., McNab, R.J., Boddie, S.J., and Bainbridge, D. (2000) Greenstone: A comprehensive open-source digital library software system *Proceedings of the Fifth ACM Conference on Digital Libraries 2000.* (2000) ACM Press, 113-121.