f i ® s t m ¤ ñ d @ ¥
PEER-REVIEWED JOURNAL ON THE INTERNET

The Usability of Open Source Software

by David M. Nichols and Michael B. Twidale

Abstract

The Usability of Open Source Software by David M. Nichols and Michael B. Twidale
Open source communities have successfully developed a great deal of software although most computer users only use proprietary applications. The usability of open source software is often regarded as one reason for this limited distribution. In this paper we review the existing evidence of the usability of open source software and discuss how the characteristics of open source development influence usability. We describe how existing human-computer interaction techniques can be used to leverage distributed networked communities, of developers and users, to address issues of usability.

Contents

Introduction

Open source communities have successfully developed a great deal of software. Most of this software is used by technically sophisticated users, in software development or as part of the larger computing infrastructure. Although the use of open source software is growing, the average user computer user only directly interacts with proprietary software. There are many reasons for this situation; one of which is the perception that open source software is less usable. This paper examines how the open source development process influences usability and suggests usability improvement methods that are appropriate for community-based software development on the Internet.

One interpretation of this topic can be presented as the meeting of two different paradigms:

- the open source developer-user who both uses the software and contributes to its development
- the user-centred design movement that attempts to bridge the gap between programmers and users through specific techniques (usability engineering, participatory design, ethnography etc.)

Indeed the whole rationale behind the user-centred design approach within human-computer interaction (HCI) emphasises that software developers cannot easily design for typical users. At first glance this suggests that open source developer communities will not easily live up to the goal of replacing proprietary software on the desktop of most users (Raymond, 1998). However, as we discuss in this paper, the situation is more complex and there are a variety of potential approaches: attitudinal, practical and technological.

In this paper we first review the existing evidence of the usability of open source software (OSS). We then outline the ways in which the characteristics of open source development influence the software. Finally, we describe how existing HCI techniques can be used to leverage distributed networked communities to address issues of usability.

Is there an open source usability problem?

Open source software has gained a reputation for reliability, efficiency, functionality that has surprised many people in the software engineering world. The Internet has facilitated the coordination of volunteer developers around the world to produce open source solutions that are market leaders in their sector (e.g. the Apache Web server). However most of the users of these applications are relatively technically sophisticated and the average desktop user is using standard commercial proprietary software (Lerner and Tirole, 2002). There are several explanations for this situation: inertia, interoperability, interacting with existing data, user support, organisational purchasing decisions etc. In this paper we are concerned with one possible explanation: that (for most potential users) open source software has poorer usability.

Usability is typically described in terms of five characteristics: ease of learning, efficiency of use, memorability, error frequency and severity, and subjective satisfaction (Nielsen, 1993). Usability is separate from the utility of software (whether it can perform some function) and from other characteristics such as reliability and cost. Software, such as compilers and source code editors, which is used by developers does not appear to represent a significant usability problem for OSS. In the following discussion we concentrate on software (such as word processors, e-mail clients and Web browsers) which is aimed predominantly at the average user.

That there are usability problems with open source software is not significant by itself; all interactive software has problems. The issue is: how does software produced by an open source development process compare with other approaches? Unfortunately it is not easy to arrange a controlled experiment to compare the alternative engineering approaches; however it is possible to compare similar tasks on existing software programs produced in different

development environments. The only study we are aware of that does such a comparison is Eklund et al. (2002), using Microsoft Excel and StarOffice (this particular comparison is made more problematic by StarOffice's proprietary past).

There are many differences between the two programs that may influence such comparisons, e.g. development time, development resources, maturity of the software, prior existence of similar software etc. Some of these are factors are characteristic of the differences between open source and commercial development but the large number of differences make it difficult to determine what a 'fair comparison' should be. Ultimately user testing of the software, as with Eklund et al. (2002), must be the acid test. However, as has been shown by the Mozilla Project (Mozilla, 2002), it may take several years for an open source project to reach comparability and premature negative comparisons should not be taken as indicative of the whole approach. Additionally, the public nature of open source development means that the early versions are visible, whereas the distribution of embryonic commercial software is usually restricted.

There is a scarcity of published usability studies of open source software, in addition to Eklund et al. (2002) we are aware only of studies on GNOME (Smith et al., 2001), Athena (Athena, 2001) and Greenstone (Nichols et al., 2001). The characteristics of open source projects emphasize continual incremental development that does not lend itself to traditional formal experimental studies (although culture may play a part as we discuss in the next section).

Although there are few formal studies of open source usability there are several suggestions that open source software usability is a significant issue (Behlendorf, 1999; Raymond, 1999; Manes, 2002; Nichols et al., 2001; Thomas, 2002; Frishberg et al., 2002):

"If this [desktop and application design] were primarily a technical problem, the outcome would hardly be in doubt. But it isn't; it's a problem in ergonomic design and interface psychology, and hackers have historically been poor at it. That is, while hackers can be very good at designing interfaces for other hackers, they tend to be poor at modeling the thought processes of the other 95% of the population well enough to write interfaces that J. Random End-User and his Aunt Tillie will pay to buy." (Raymond, 1999)

"Traditionally the users of OSS have been experts, early adopters, and nearly synonymous with the development pool. As OSS enters the commercial mainstream, a new emphasis is being placed on usability and interface design, with Caldera and Corel explicitly targeting the general desktop with their Linux distributions. Non-expert users are unlikely to be attracted by the available source code and more likely to choose OSS products on the basis of cost, quality, brand and support." (Feller and Fitzgerald, 2000)

Raymond is stating the central message of user-centred design (Norman and Draper, 1986): developers need specific external help to cater for the average user. The HCI community has developed several tools and techniques for this purpose: usability inspection methods, interface guidelines, testing methods, participatory design, inter-disciplinary teams etc. (Nielsen, 1993). The increasing attention being paid to usability in open source circles (Frishberg et al., 2002) suggests that it may be passing through a similar phase to that of proprietary software in the 1980s.

As the users of software became more heterogeneous, and less technically experienced, software producers started to adopt user-centred methods to ensure that their products were successfully adopted by their new users. Whilst many users continue to have problems with software applications, the HCI specialists employed by companies have greatly improved users' experiences.

As the user base of OSS widens to include many non-developers, projects will need to apply HCI techniques if they wish their software to be used on the desktop of the average user. There is recent evidence (Benson et al., 2002; Biot, 2002) that some open source projects are adopting techniques from previous proprietary work, such as explicit user interface guidelines for application developers (Benson et al., 2002).

It is difficult to give a definitive answer to the question: is there an open source usability problem? The existence of a problem does not necessarily mean that all OSS interfaces are bad or that OSS is doomed to have hard to use interfaces, just a recognition that the interfaces ought to be and can be made better. The opinions of several commentators and the actions of companies, such as Sun's involvement with GNOME, are strongly suggestive that there is a problem, although the academic literature (e.g. Feller and Fitzgerald, 2002) is largely silent on the issue (Frishberg et al. (2002) and Nichols et al. (2001) are the main exceptions). However, in order to suggest HCI approaches that mesh with the practical and social characteristics of open source developers (and users) it is necessary to examine the aspects of the development process that may have a negative impact on usability.

Usability and open source software development

"They just don't like to do the boring stuff for the stupid people!" (Sterling, 2002)

To understand the usability of current OSS we need to examine the current software development process. It is a truism of user-centred design that the development activities are reflected in the developed system. Drawing extensively from two main sources (Nichols et al., 2001; Thomas, 2002), we present here a set of features of the OSS development process that appear to contribute to the problem of poor usability. In addition there are some features that are shared with the commercial sector that help to explain why if OSS usability is no worse than proprietary systems, nor is it any better.

This list of features is not intended to be complete but to serve as a starting point in addressing these issues. We note that there would seem to be significant difficulties in 'proving' whether several of these hypotheses are correct.

Developers are not typical end-users

This is a key point of Nielsen (1993), and is one shared with commercial systems developers. Teaching computer science students about usability issues is, in our experience, chiefly about helping them to try and see the use of their systems through the eyes of other people unlike

themselves and their peers. In fact, for many more advanced OSS products, developers are indeed users, and these esoteric products with interfaces that would be unusable by a less technically skilled group of users are perfectly adequate for their intended elite audience. Indeed there may be a certain pride in the creation of a sophisticated product with a powerful, but challenging to learn interface. Mastery of such a product is difficult and so legitimates membership of an elite who can then distinguish itself from so-called 'lusers' [1]. Trudelle (2002) comments that "the product [a Web browser] should target people whom they [OSS contributors] consider to be clueless newbies."

However when designing products for less technical users, all the traditional usability problems arise. In the Greenstone study (Nichols et al., 2001) common command line conventions, such as a successful command giving no feedback, confused users. The use of the terms 'man' (from the Unix command line), when referring to the help system, and 'regexp' (regular expression) in the GNOME interface are typical examples of developer terminology presented to end-users (Smith et al., 2001).

The OSS approach fails for end user usability because there are 'the wrong kind of eyeballs' looking at, but failing to see, usability issues. In some ways the relatively new problem with OSS usability reflects the earlier problem with commercial systems development: initially the bulk of applications were designed by computing experts for other computing experts, but over time an increasing proportion of systems development was aimed at non-experts and usability problems became more prominent. The transition to non-expert applications in OSS products is following a similar trajectory, just a few years later.

The key difference between the two approaches is this: commercial software development has recognised these problems and can employ specific HCI experts to 're-balance' their historic team compositions and consequent development priorities in favour of users (Frishberg et al., 2002). However, volunteer-led software development does not have the ability to hire in missing skill sets to ensure that user-centred design expertise is present in the development team. Additionally, in commercial development it is easier to ensure that HCI experts are given the sufficient authority to promote the interests of users.

Usability experts do not get involved in OSS projects

Anecdotal evidence suggests that few people with usability experience are involved in OSS projects; one of the 'lessons learned' in the Mozilla project (Mozilla, 2002) is to "ensure that UI [user interface] designers engage the Open Source community" (Trudelle, 2002). Open source draws its origins and strength from a hacker culture (O'Reilly, 1999). This culture can be extremely welcoming to other hackers, comfortably spanning nations, organisations and time zones via the Internet. However it may be less welcoming to non-hackers.

Good usability design draws from a variety of different intellectual cultures including but not limited to psychology, sociology, graphic design and even theatre studies. Multidisciplinary design teams can be very effective, but require particular skills to initiate and sustain. As a result, existing OSS teams may just lack the skills to solve usability problems and even the skills to bring in 'outsiders' to help. The stereotypes of low hacker social skills are not to be taken as gospel, but the sustaining of distributed multidisciplinary design teams is not trivial.

Furthermore, the skills and attitudes necessary to be a successful and productive member of an OSS project may be relatively rare. With a large candidate set of hacker-programmers interested in getting involved, OSS projects have various methods for winnowing out those with the best skill sets and giving them progressively more control and responsibility. It may be that the same applies to potential usability participants, implying that a substantial number of potential usability recruits are needed in order to proceed with the winnowing process. If true, this adds to the usability expertise shortage problem.

There are several possible explanations for the minimal or non-participation of HCI and usability people in OSS projects:

- There are far fewer usability experts than hackers, so there are just not enough to go around.
- Usability experts are not interested in, or incentivised by the OSS approach in the way that many hackers are.
- Usability experts do not feel welcomed into OSS projects.
- Inertia: traditionally projects haven't needed usability experts. The current situation of many technically adept programmers and few usability experts in OSS projects is just an historical artifact.
- There is not a critical mass of usability experts involved for the incentives of peer acclaim and recruitment opportunities to operate.

The incentives in OSS work better for improvement of functionality than usability

Are OSS developers just not interested in designing better interfaces? As most work on open source projects is voluntary, developers work on the topics that interest them and this may well not include features for novice users. The importance of incentives in OSS participation is well recognised (Feller and Fitzgerald, 2002; Hars and Ou, 2001). These include the gaining of respect from peers and the intrinsic challenge of tackling a hard problem. Adding functionality or optimising code provide opportunities for showing off one's talents as a hacker to other hackers. If OSS participants perceive improvements to usability as less high status, less challenging or just less interesting, then they are less likely to choose to work on this area. The voluntary nature of participation has two aspects: choosing to participate at all and choosing which out of usually a large number of problems within a project to work on. With many competing challenges, usability problems may get crowded out.

An even more extreme version of this case is that the choice of the remit of an entire OSS project may be more biased towards the systems side than the applications side [2]. "Almost all of the most widely-known and successful OSS projects seem to have been initiated by someone who had a technical need that was not being addressed by available proprietary (or OSS) technology" [3]. Raymond refers to the motivation of "scratching a personal itch" (Raymond, 1998). Clearly the technically adept initiators of OSS projects are more likely to have a personal need for very advanced applications, development toolkits or systems infrastructure improvements than an application that also happens to meet the needs of a less technically sophisticated user.

"From a developer's perspective, solving a usability problem might not be a rewarding experience since the solution might not involve a programming challenge, new technology or algorithms. Also the benefits from solving the usability problem might be a slight change to the behavior of the software (even though it might cause a dramatic improvement from user's perspective). This change in behavior might be subtle, and not fit into the typical types of contributions developers make such as adding features, or bug fixes." (Eklund et al., 2002)

The 'personal itch' motivation creates a significant difference between open source and commercial software development. Commercial systems development is usually about solving the needs of another group of users. The incentive is to make money by selling software to customers, often customers who are prepared to pay precisely because they do not have the development skills themselves. Capturing the requirements of software for such customers is acknowledged as a difficult problem in software engineering and consequently techniques have been developed to attempt to address it. By contrast, many OSS projects lack formal requirements capture processes and even formal specifications (Scacchi, 2002). Instead they rely on understood requirements of initially individuals or tight-knit communities. These are supported by 'informalisms' and illustrated by the evolving OSS project code that embodies, even if it does not articulate, the requirements.

The relation to usability is that this implies that OSS is in certain ironic ways more egotistical than closed source software (CSS). A personal itch implies designing software for one's own needs. Explicit requirements are consequently less necessary. Within OSS this is then shared with a like-minded community and the individual tool is refined and improved for the benefit of all — within that community. By contrast, a CSS project may be designing for use by a community with different characteristics, and where there is a strong incentive to devote resources to certain aspects of usability, particularly initial learnability, in order to maximise sales (Varian, 1993).

Usability problems are harder to specify and distribute than functionality problems

Functionality problems are easier to specify, evaluate and modularize than certain usability problems. These are all attributes which simplify decentralized problem solving. Some (but not all) usability problems are much harder to describe and may pervade an entire screen, interaction or user experience. Incremental patches to interface bugs may be far less effective than incremental patches to functionality bugs. Fixing the problem may require a major overhaul of the entire interface — clearly not a small contribution to the ongoing design work. Involving more than one designer in interface design, particularly if they work autonomously, will lead to design inconsistency and hence lower the overall usability. Similarly, improving an interface aspect of one part of the application may require careful consideration of the consequences of that change for overall design consistency. This can be contrasted with the incremental fixing of the functionality of a high quality modularised application. The whole point of modularisation is that the effects are local. Substantial (and highly desirable) refactoring can occur throughout the ongoing project while remaining invisible to the users. However, many interface changes are global in scope because of their consistency effects.

The modularity of OSS projects contributes to the effectiveness of the approach (O'Reilly, 1999), enabling them to side-step Brooks' Law [4]. Different parts can be swapped out and

replaced by superior modules that are then incorporated in the next version. However a major success criterion for usability is consistency of design. Slight variations in the interface between modules and different versions of modules can irritate and confuse, marring the overall user experience. Their inevitably public nature means that interfaces are not amenable to the black-boxing that permits certain kinds of incremental and distributed improvement.

We must note that OSS projects do successfully address certain categories of usability problems. One popular approach to OSS interface design is the creation of 'skins': alternate interface layouts which dramatically affect the overall appearance of the application, but do little to change the nature of the underlying interaction dynamics. A related approach is software internationalization, where the language of the interface (and any culture-specific icons) is translated. Both approaches are amenable to the modular OSS approach whereas an attempt to address deeper interaction problems by a redesign of sets of interaction sequences does not break down so easily into a manageable project. The reason for the difference is that addressing the deeper interaction problems can have implications across not only the whole interface, but also lead to requirements for changes to different elements of functionality.

Another major category of OSS usability success is in software (chiefly GNU/Linux) installation. Even the technically adept had difficulties in installing the early versions of GNU/Linux. The Debian project (Debian, 2002) was initiated as a way to create a better distribution that made installation easier, and other projects and companies have continued this trend. Such projects solve a usability problem, but in a manner that is compatible with traditional OSS development. Effectively a complex set of manual operations is automated, creating a black box for the end user with no wish to explore further. Of course since it is an open source project, the black box is openable, examinable and changeable for those with the will and the skill to investigate.

Design for usability really ought to take place in advance of any coding

In some ways it is surprising that OSS development is so successful, given that it breaks many established rules of conventional software engineering. Well run projects are meant to plan carefully in advance, capturing requirements and clearly specifying what should be done before ever beginning coding. By contrast OSS often appears to involve coding as early as possible, relying on constant review to refine and improve the overall, emergent design: "your nascent developer community needs to have something runnable and testable to play with" (Raymond, 1998). Similarly, Scacchi's (2002) study didn't find "examples of formal requirements elicitation, analysis and specification activity of the kind suggested by software engineering textbooks." Trudelle (2002) notes that skipping much of the design stage with Mozilla resulted in design and requirements work occurring in bug reports, after the distribution of early versions.

This approach does seem to work for certain kinds of applications, and in others there may be a clear plan or shared vision between the project coordinator and the main participants. However good interface design works best by being involved before coding occurs. If there is no collective advance planning even for the coding, there is no opportunity to factor in interface issues in the early design. OSS planning is usually done by the project initiator, before the larger group are involved [5]. We speculate that while an OSS project's members may share a strong sense of vision of the intended functionality (which is what allows the

bypassing of traditional software engineering advance planning), they often have a much weaker shared vision of the intended interface. Unless the initiator happens to possess significant interaction design skills, important aspects of usability will get overlooked until it is too late. As with many of the issues we raise, that is not to say that CSS always, or even frequently, gets it right. Rather we want to consider potential barriers within existing OSS practice that might then be addressed.

Open source projects lack the resources to undertake high quality usability work

OSS projects are voluntary and so work on small budgets. Employing outside experts such as technical authors and graphic designers is not possible. As noted earlier there may currently be barriers to bring in such skills within the volunteerist OSS development team. Usability laboratories and detailed large scale experiments are just not economically viable for most OSS projects. Discussion on the K Desktop Environment (KDE) Usability (KDE Usability, 2002) mailing list has considered asking usability laboratories for donations of time in which to run studies with state of the art equipment.

Recent usability activity in several open source projects has been associated with the involvement of companies, e.g. Benson et al. (2002), although it seems likely that they are investing less than large proprietary software developers. Unless OSS usability resources are increased, or alternative approaches are investigated (see below), then open source usability will continue to be constrained by resource limitations.

Commercial software establishes state of the art so that OSS can only play catch-up

Regardless of whether commercial software provides good usability, its overwhelming prominence in end user applications creates a distinct inertia with respect to innovative interface design. In order to compete for adoption, OSS applications appear to follow the interface ideas of the brand leaders. Thus the Star Office spreadsheet component, Calc, tested against Microsoft Excel in (Eklund et al., 2002) was deliberately developed to provide a similar interface in order to make transfer learning easier. As a result it had to follow the interface design ideas of Excel regardless of whether or not they could have been improved upon.

There does not seem to be any overriding reason why this conservatism should be the case, other than the perceived need to compete by enticing existing CSS users to switch to open source direct equivalents. Another possibility is that current typical OSS developers, who may be extremely supportive of functionality innovation, just lack an interest in interface design innovation. Finally, the underlying code of a commercial system is proprietary and hidden, requiring any OSS rival to do a form of reverse engineering to develop. This activity can inspire significant innovation and extension. By contrast, the system's interface is a very visible pre-existing solution which might dampen innovation — why not just copy it, subject to minor modifications due to concerns of copyright? One might expect in the absence of other factors that open source projects would be much more creative and risk-taking in their

development of radically new combinations of functionality and interface, since they do not suffer short-termist financial pressures.

OSS has an even greater tendency towards software bloat than commercial software

Many kinds of commercial software have been criticised for bloated code, consuming ever greater amounts of memory and numbers of processor cycles with successive software version releases. There is a commercial pressure to increase functionality and so to entice existing owners to purchase the latest upgrade. Naturally the growth of functionality can seriously degrade usability as the increasing number of options becomes ever more bewildering, serving to obscure the tiny subset of features that a given user wishes to employ.

There are similar pressures in open source development, but due to different causes. Given the interests and incentives of developers, there is a strong incentive to add functionality and almost no incentive to delete functionality, especially as this can irritate the person who developed the functionality in question. Worse, given that peer esteem is a crucial incentive for participation, deletion of functionality in the interest of benefiting the end user creates a strong disincentive to future participation, perhaps considered worse than having one's code replaced by code that one's peers have deemed superior. The project maintainer, in order to keep volunteer participants happy, is likely to keep functionality even if it is confusing, and on receipt of two similar additional functionalities, keep both, creating options for the user of the software to configure the application to use the one that best fits their needs. In this way as many contributors as possible can gain clear credit for directly contributing to the application. This suggested tendency to 'pork barrel' design compromise needs further study.

The process of 'release early and release often' can lead to an acceptance of certain clumsy features. People invest time and effort in learning them and create their own workarounds to cope with them. When a new, improved version is released with a better interface, there is a temptation for those early adopters of the application to refuse to adapt to the new interface. Even if it is easier to learn and use than the old one, their learning of the old version is now a sunk investment and understandably they may be unwilling to re-learn and modify their workarounds. The temptation for the project maintainer is to keep multiple legacy interfaces coordinated with the latest version. This pleases the older users, creates more opportunities for development, keeps the contributions of the older interfaces in the latest version, and adds to the complexity of the final product.

OSS development is inclined to promote power over simplicity

'Software bloat' is widely agreed to be a negative attribute. However, the decision to add multiple alternative options to a system may be seen as a positive good rather than an invidious compromise. We speculate that freedom of choice may be considered a desirable attribute (even a design aesthetic) by many OSS developers. The end result is an application that has many configuration options, allowing very sophisticated tailoring by expert users, but which can be bewildering to a novice [6]. The provision of five different desktop clocks in GNOME (Smith et al., 2001) is one manifestation of this tendency; another is the growth of preferences interfaces in many OSS programs (Thomas, 2002).

Thus there is a tendency for OSS applications to grow in complexity, reducing their usability for novices, but with that tendency to remain invisible to the developers who are not novices and relish the power of sophisticated applications. Expert developers will also rarely encounter the default settings of a multiplicity of options and so are unlikely to give much attention to their careful selection, whereas novices will often live with those defaults. Of course commercial applications also grow in complexity, but at least there are some factors to moderate that growth, including the cost of developing the extra features and some pressures from a growing awareness of usability issues.

Potential approaches to improving OSS usability

The above factors aim to account for the current relatively poor state of the usability of many open source products. However there are factors that should contribute to better usability, although they may currently be outweighed by the negative factors in many current projects.

A key positive factor is that some end users are involved in OSS projects [7]. This involvement can be in elaborating requirements, testing, writing documentation, reporting bugs, requesting new features, etc. This is clearly in accord with the advocacy of HCI experts, e.g. Shneiderman (2002), and also has features in common with participatory design (Kyng and Mathiassen, 1997). The challenge is how to enable and encourage far greater participation of non-technical end users and HCI experts who do not conform to the traditional OSS-hacker stereotype.

We describe several areas where we see potential for improving usability processes in OSS development.

Commercial approaches

One method is to take a successful OSS project with powerful functionality and involve companies in the development of a better interface. It is noticeable that several of the positive (from the HCI point of view) recent developments (Smith et al., 2001; Benson et al., 2002; Trudelle, 2002) in OSS development parallel the involvement of large companies with both design experience and considerably more resources than the typical volunteer-led open source project. However the HCI methods used are basically the same as for proprietary software and do not leverage the distributed community that gives open source software its perceived edge in other aspects of development. Does this imply that the only way to achieve high level of end-user usability is to 'wrap' an open source project with a commercially developed interface? Certainly that is one approach, and the Apple OS X serves as a prime example, as to a lesser extent do commercial releases of GNU/Linux (since they are aimed at a (slightly) less technologically sophisticated market). The Netscape/Mozilla model of mutually informed development offers another model. However as Trudelle (2002) notes, there can be conflicts of interest and mutual misunderstandings between a commercial partner and OSS developers about the direction of interface development so that it aligns with their interests.

Technological approaches

One approach to dealing with a lack of human HCI expertise is to automate the evaluation of interfaces. Ivory and Hearst (2001) present a comprehensive review of automated usability evaluation techniques and note several advantages to automation including cost reduction, increased consistency and the reduced need for human evaluators. For example, the Sherlock tool (Mahjan and Shneiderman, 1997) automated checking visual and textual consistency across an application using simple methods such as a concordance of all text in the application interface and metrics such as widget density. Applications with interfaces that can be easily separated from the rest of the code, such as Mozilla, are good candidates for such approaches.

An interesting approach to understanding user behaviour is the use of 'expectation agents' (Hilbert and Redmiles, 2001) that allow developers to easily explicitly place their design expectations as part of an application. When a user does something unexpected (and triggers the expectation agent) program state information is collected and sent back to the developers. This is an extension of the instrumentation of applications but one that is focussed on user activity (such as the order in which a user fills in a form) rather than the values of program variables. Extensive instrumentation has been used by closed source developers as a key element of program improvement (Cusmano and Selby, 1995).

Academic involvement

It is noticeable some of the work described earlier has emerged from higher education (Athena, 2001; Eklund et al., 2002; Nichols et al., 2001). In these cases classes of students involved in HCI have participated in or organised studies of OSS. This type of activity is effectively a gift to the software developers, although the main aim is pedagogical. The desirability of practising skills and testing conceptual understanding on authentic problems rather than made-up exercises are obvious.

The model proposed is that an individual, group or class would volunteer support following the OSS model, but involving aspects of any combination of usability analysis and design: user studies, workplace studies, design requirements, controlled experiments, formal analysis, design sketches, prototypes or actual code suggestions. In order to support these kinds of participation, certain changes may be needed to the OSS support software, as noted below.

Involving the end users

The Mozilla bug database, Bugzilla, has received more then 150,000 bug reports at the time of writing. Overwhelmingly these bugs reports concern functionality (rather than usability) and have been contributed by technically sophisticated users and developers:

"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." [8]

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 (Nakakoji, 2002).

One reason for users' non-participation is that the act of contributing is perceived as too costly compared to any benefits. The time and effort to register with Bugzilla (a pre-requisite for bug reporting) and understand its Web interface are considerable. The language and culture embodied in the tool are themselves barriers to participation for many users. In contrast the crash reporting tools in both Mozilla and Microsoft Windows XP are simple to use and require no registration. Furthermore, these tools are part of the application and do not require a user to separately enter information at a Web site.

We suggest that integrated user-reported usability incidents are a strong candidate for addressing usability issues in OSS projects. That is, users reporting occasions when they have problems whilst they are using an application. Existing HCI research (Hartson and Castillo, 1998; Castillo et al., 1998; Thompson and Williges, 2000) has shown, on a small scale, that user reporting is effective at identifying usability problems. These reporting tools are part of an application, easy to use, free of technical vocabulary and can return objective program state information in addition to user comments (Hilbert and Redmiles, 2000). This combination of objective and subjective data is necessary to make causal inferences about users' interactions in remote usability techniques (Kaasgaard et al., 1999). In addition to these user-initiated reports, applications can prompt users to contribute based on their behaviour (Ivory and Hearst, 2001). Kaasgaard et al. (1999) note that it is hard to predict how these additional functionalities affect the main usage of the system.

Another method to involve users is to create packaged remote usability tests that can be performed by anyone at any time. The results are collated on the user's computer and sent back to the developers. Tullis et al. (2002) and Winckler et al. (1999) both describe this approach for usability testing of Web sites; a separate browser window is used to guide a user through a sequence of tasks in the main window. Scholtz (1999) describes a similar method for Web sites within the main browser window — effectively as part of the application. Comparisons of laboratory-based and remote studies of Web sites indicate that users' task completion rates are similar and that the larger number of remote participants compensates for the lack of direct user observation (Tullis et al., 2002; Jacques and Savastano, 2001).

Both of these approaches allow users to contribute to usability activities without learning technical vocabulary. They also map well onto the OSS approach: they allow participation according to the contributor's expertise and leverage the strengths of a community in a distributed networked environment. Although these techniques lose the control of laboratory-based usability studies they gain in authenticity in that they are firmly grounded in the user's environment (Thomas and Kellogg, 1989; Jacques and Savastano, 2001; Thomas and Macredie, 2002).

To further promote user involvement it should be possible to easily track the consequences of a report, or test result, that a user has contributed. The public nature of Bugzilla bug discussions achieves this for developers but a simpler version would be needed for average users so that they are not overwhelmed by low-level detail. Shneiderman [9] suggests that users might be financially rewarded for such contributions, such as discounts on future software purchases. However in an open source context the user could expect information such as: "Your four recent reports have contributed to the fixing of bug X which is reflected in the new version 1.2.1 of the software."

Creating a usability discussion infrastructure

For functional bugs a tool such as Bugzilla works well in supporting developers, but presents complex interfaces to other potential contributors. If we wish such tools to be used by HCI people then they may need an alternative lightweight interface that abstracts away from some low-level details. In particular, systems that are built on top of code management systems can easily become overly focussed on textual elements.

As user reports and usability test results are received they need to structured, analysed, discussed and acted on. Much usability discussion is graphical in nature and might be better supported through sketching and annotation functionality; it is noticeable that some Mozilla bug discussions include *textual* representations ('ASCII art') of proposed *interface* elements. Hartson and Castillo (1998) review various graphical approaches to bug reporting including video and screenshots which can supplement the predominant text-based methods. For example, an application could optionally include a screenshot with a bug report; the resulting image could then be annotated as part of an online discussion. Although these may seem like minor changes, a key lesson of usability research is that details matter and that a small amount of extra effort is enough to deter users from participating (Nielsen, 1993).

Fragmenting usability analysis and design

We can envisage various new kinds of lightweight usability participation, that can be contrasted with the more substantial experimental and analysis contributions outlined above for academic or commercial involvement. An end user can volunteer a description of their own, perhaps idiosyncratic, experiences with the software. A person with some experience of usability can submit their analysis. Furthermore, such a contributor could run a user study with a sample size of one, and then report it. It is often surprising how much usability information can be extracted from a small number of studies (Nielsen, 1993).

In the same way that OSS development work succeeds by fragmenting the development task into manageable sub-units, so usability testing can be fragmented by involving many people worldwide each doing a single user study and then combining the overall results for analysis. Coordinating many parallel small studies would require tailored software support but it opens up a new way of undertaking usability work that maps well onto the distributed nature of OSS development. Work on remote usability (Hartson et al., 1996; Scholtz, 2001) strongly suggests that the necessary *distribution* of work is feasible; further work is needed in coordinating and interpreting the results.

Involving the experts

A key point for involving HCI experts will be a consideration of the incentives of participation. We have noted the issues of a critical mass of peers, and a legitimisation of the importance of usability issues within the OSS community so that design trade-offs can be productively discussed. One relatively minor issue is the lowering of the costs of participation caused by problems with articulating usability in a predominantly textual medium, and various solutions have been proposed. We speculate that for some usability experts, their participation in an OSS project will be problematic in cases where their proposed designs and improvements clash with the work of traditional functionality-centric development. How can

this be resolved? Clear articulation of the underlying usability analysis, a kind of design rationale, may help. In the absence of such explanations, the danger is that a lone usability expert will be marginalized.

Another kind of role for a usability expert can be as the advocate of the end user. This can involve analysing end user contributions, creating a condensed version, perhaps filtered by the expert's own theoretical understanding to address concerns of developers that the reports are biased or unrepresentative. The expert then engages in the design debate on behalf of the end users, attempting to rectify the problem of traditional OSS development only scratching the personal itches of the developers, not of intended users. As with creating incentives to promote the involvement of end users, the consequences on the evolving design of usability experts' interactions should be recorded and be easily traceable.

Education and evangelism

In the same way that commercial software development organisations had to learn that usability was an important issue that they should consider, and which could have a significant impact on the sales of their product, so open source projects will need to be convinced that it is worth the effort of learning about and putting into practice good usability development techniques. The incentive of greater sales will not usually be relevant, and so other approaches to making the usability case will need to be made. Nickell (2001) suggests that developers prefer that their programs are used and that "most hackers find gaining a userbase to be a motivating factor in developing applications."

Creating a technological infrastructure to make it easier for usability experts and end users to participate will be insufficient without an equivalent social infrastructure. These new entrants to OSS projects will need to feel welcomed and valued, even if (actually because) they lack the technical skills of traditional hackers. References to 'clueless newbies' and 'lusers', and some of the more vituperative online arguments will need to be curtailed, and if not eliminated, at least moved to designated technology-specific areas. Beyond merely tolerating a greater diversification of the development team, it would be interesting to explore the consequences of certain OSS projects actively soliciting help from these groups. As with various multidisciplinary endeavours, including integrating psychologists into commercial interface design and ethnographers into computer supported cooperative work projects, care needs to be taken in enabling the participants to be able to talk productively to each other (Crabtree et al., 2000).

Discussion and future work

We do not want to imply that OSS development has completely ignored the importance of good usability. Recent activities (Frishberg et al., 2002; Nickell, 2001; Smith et al., 2001) suggest that the open source community is increasing its awareness of usability issues. This paper has identified certain barriers to usability and explored how these are being and can be addressed. Several of the approaches outlined above directly mirror the problems identified

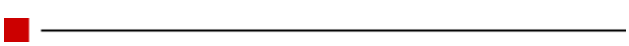
earlier; try automated evaluation where there is a shortage of human expertise and encourage various kinds of end user and usability expert participation to re-balance the development community in favour of average users. If traditional OSS development is about scratching a personal itch, usability is about being aware of and concerned about the itches of others.

Deeper investigation of the issues outlined in this paper could take various forms. One of the great advantages of OSS development is that its process is to a large extent visible and recorded. A study of the archives of projects (particularly those with a strong interface design component such as GNOME, KDE and Mozilla) will enable a verification of the claims and hypotheses ventured here, as well as the uncovering of a richer understanding of the nature of current usability discussions and development work. Example questions include: 'How do HCI experts successfully participate in OSS projects?', 'Do certain types of usability issues figure disproportionately in discussions and development effort?' and 'What distinguishes OSS projects that are especially innovative in their functionality and interface designs?'

The approaches outlined in the previous section need further investigation and indeed experimentation to see if they can be feasibly used in OSS projects, without disrupting the factors that make traditional functionality-centric OSS development so effective. These approaches are not necessarily restricted to OSS; several can be applied to proprietary software. Indeed the ideas derived from discount usability engineering and participatory design originated in developing better proprietary software. However, they may be even more appropriate for open source development in that they map well on to the strengths of a volunteer developer community with open discussion.

Most HCI research has concentrated on pre-release activities that inform design and relatively little on post-release techniques (Hartson and Castillo, 1998; Smilowitz et al., 1994). It is noteworthy that participatory design is a field in its own right whereas participative usage is usually quickly passed over by HCI textbooks. Thus OSS development in this case need not merely play catch-up with the greater end user focus of the commercial world, but potentially can innovate in exploring how to involve end users in subsequent redesign. There have been several calls in the literature (Shneiderman 2002; Lieberman and Fry, 2001; Fischer, 1998) for users to become actively involved in software development beyond standard user-centred design activities (such as usability testing, prototype evaluation and fieldwork observation). It is noticeable that these comments seem to ignore that this involvement is already happening in OSS projects.

Raymond (1998) comments that "debugging is parallelizable", we can add to this that usability reporting, analysis and testing is also parallelisable. However certain aspects of usability design do not appear to be so easily parallelisable. We believe that these issues should be the focus of future research and development, understanding how they have operated in successful projects and designing and testing technological and organisational mechanisms to enhance future parallelisation. In particular, future work should seek to examine whether the issues identified in this paper are historical in nature (i.e. they flow from the particular ancestry of open source development) or are necessarily connected to the OSS development model.



Conclusion

Improvements in the usability of open source software do not necessarily mean that such software will displace proprietary software from the desktop; there are many other factors involved, e.g. inertia, support, legislation, legacy systems etc. However improved usability is a necessary condition for such a spread. We believe this paper is the first detailed discussion of these issues in the literature.

Lieberman and Fry (2001) foresee that "interacting with buggy software will be a cooperative problem-solving activity of the end user, the system, and the developer." For some open source developers this is already true, expanding this situation to (potentially) include all of the end-users of the system would mark a significant change in software development practices.

There are many techniques from HCI that can be easily and cheaply adopted by open source developers. Additionally there are several approaches that seem to provide a particularly good fit with a distributed networked community of users and developers. If open source projects can provide a simple framework for users to contribute non-technical information about software to the developers then they can leverage and promote the participatory ethos amongst their users.

Raymond (1998) proposed that "given enough eyeballs all bugs are shallow." For seeing usability bugs, the traditional open source community may comprise the wrong kind of eyeballs. However it may be that by encouraging greater involvement of usability experts and end users it is the case that: given enough user experience reports all usability issues are shallow. By further engaging typical users into the development process OSS projects can create a networked development community that can do for usability what it has already done for functionality and reliability.

About the Authors

David M. Nichols is a lecturer in the Department of Computer Science at the University of Waikato, Hamilton, New Zealand.
E-mail: dmn@cs.waikato.ac.nz
Direct comments to dmn@cs.waikato.ac.nz

Michael B. Twidale is an associate professor in the Graduate School of Library and Information Science (GSLIS) at the University of Illinois at Urbana-Champaign.
E-mail: twidale@uiuc.edu

Acknowledgements

We would like to thank several people for valuable discussions and comments about this topic: Damon Chaplin, Dave Dubin, Ian Murdock, Havoc Pennington, Walt Scacchi, Matthew Thomas, Stuart Yeates, the GSLIS writing group at UIUC, the HCI Group in the Department of Computer Science at the University of Waikato and many people who commented via Slashdot.

Notes

1. Raymond and Steele, 1991, p. 364.

2. Feller and Fitzgerald, 2002, p. 114.

3. Feller and Fitzgerald, 2002, p. 139.

4. Feller and Fitzgerald, 2002, p. 85.

5. Feller and Fitzgerald, 2002, p. 101.

6. Nielsen, 1993, p. 12.

7. Feller and Fitzgerald, 2002, p. 93; Raymond, 1998.

8. A Bugzilla comment at http://bugzilla.mozilla.org/show_bug.cgi?id=89907#c14.

9. Shneiderman, 2002, p. 29.

References

Athena, 2001. "Usability Testing of Athena User Interface," MIT Information Systems, at http://web.mit.edu/is/usability/aui/, accessed 28 November 2002.

Brian Behlendorf, 1999. "Open source as a business strategy," In: M. Stone, S. Ockman, and C. DiBona (editors). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, Calif.: O'Reilly & Associates, pp. 149-170, and at http://www.oreilly.com/catalog/opensources/book/brian.html, accessed 28 November 2002.

Calum Benson, Adam Elman, Seth Nickell, and Colin Z. Robertson, 2002. "GNOME Human Interface Guidelines 1.0," The GNOME Usability Project, at http://developer.gnome.org/projects/gup/hig/1.0/, accessed 10 October 2002.

Sebastien Biot, 2002. "KDE Usability — First Steps," KDE Usability Project, at http://usability.kde.org/activity/testing/firststeps/, accessed 28 November 2002.

José C. Castillo, H. Rex Hartson, and Deborah Hix, 1998. "Remote Usability Evaluation: Can Users Report Their Own Critical Incidents?" *Proceedings of the Conference on Human Factors on Computing Ssytems (CHI'98): Summary.* New York: ACM Press, pp. 253-254.

Andy Crabtree, David M. Nichols, Jon O'Brien, Mark Rouncefield, and Michael B. Twidale, 2000. "Ethnomethodologically Informed Ethnography and Information System Design," *Journal of the American Society for Information Science,* volume 51, number 7, pp. 666-682.

Michael A. Cusmano and Richard W. Selby, 1995. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People.* New York: The Free Press.

Debian, 2002. Debian GNU/Linux, at http://www.debian.org, accessed 28 November 2002.

Susanne Eklund, Michal Feldman, Mary Trombley, and Rashmi Sinha, 2002. "Improving the Usability of Open Source Software: Usability Testing of StarOffice Calc," presented at the *Open Source Meets Usability Workshop, Conference on Human Factors in Computer Systems (CHI 2002),* Minneapolis, Minn., at http://www.sims.berkeley.edu/~sinha/opensource.html, accessed 28 November 2002.

Joseph Feller and Brian Fitzgerald, 2002. *Understanding Open Source Software Development.* London: Addison-Wesley.

Joseph Feller and Brian Fitzgerald, 2000. "A Framework Analysis of the Open Source Software Development Paradigm," In: W.J. Orlikowski, P. Weill, S. Ang, H.C. Krcmar, and J.I. DeGross (editors). *Proceedings of the 21st Annual International Conference on Information Systems,* Brisbane, Queensland, Australia, pp. 58-69.

Gerhard Fischer, 1998. "Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments," *Automated Software Engineering,* volume 5, number 4, pp. 447-464.

Nancy Frishberg, Anna Marie Dirks, Calum Benson, Seth Nickell, and Suzanna Smith, 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: ACM Press, pp. 932-933.

Alexander Hars and Shaosong Ou, 2001. "Working for Free? — Motivations of Participating in Open Source Projects," *Proceedings of the 34th Annual Hawaii International Conference on System Sciences.* New York: IEEE Computer Society Press, pp. 2284-2292.

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

H. Rex Hartson, José C. Castillo, John Kelso, and Wayne C. Neale, 1996. "Remote Evaluation: The Network as an Extension of the Usability Laboratory," *Proceedings of the*

Conference on Human Factors in Computing Systems (CHI'96). New York: ACM Press, pp. 228-235.

David M. Hilbert and David F. Redmiles, 2001. "Large-Scale Collection of Usage Data to Inform Design," In: M. Hirose (editor). *Human-Computer Interaction — INTERACT'01: Proceedings of the Eighth IFIP Conference on Human-Computer Interaction,* Tokyo, Japan, pp. 569-576.

David M. Hilbert and David F. Redmiles, 2000. "Extracting Usability Information from User Interface Events," *ACM Computing Surveys,* volume 32, number 4, pp. 384-421.

Melody Y. Ivory and Marti A. Hearst, 2001. "The State of the Art in Automated Usability Evaluation of User Interfaces," *ACM Computing Surveys,* volume 33, number 4, pp. 470-516.

Richard Jacques and Herman Savastano, 2001. "Remote vs. Local Usability Evaluation of Web Sites," In: J. Vanderdonckt, A. Blandford, and A. Derycke (editors). *Proceedings of Joint AFIHM-BCS Conference on Human Computer Interaction (IHM-HCI'2001),* volume 2. Toulouse: Cépaduès-Editions, pp. 91-92.

KDE Usability, 2002. KDE Usability Project, at http://usability.kde.org, accessed 28 November 2002.

Morten Kyng and Lars Mathiassen (editors), 1997. *Computers and Design in Context.* Cambridge, Mass.: MIT Press.

Josh Lerner and Jean Tirole, 2002. "Some Simple Economics of Open Source," *Journal of Industrial Economics,* volume 50, number 2, pp. 197-234.

Henry Lieberman and Christopher Fry, 2001. "Will Software Ever Work?" *Communications of the ACM,* volume 44, number 3, pp. 122-124.

Rohit Mahjan and Ben Shneiderman, 1997. "Visual and Text Consistency Checking Tools for Graphical User Interfaces," *IEEE Transactions on Software Engineering,* volume 23, number 11, pp. 722-735.

Stephen Manes, 2002. "Linux Gets Friendlier," *Forbes* (10 June), pp. 134-136.

Mozilla, 2002. Mozilla Project, at http://www.mozilla.org, accessed 28 November 2002.

Kamiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye , 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, pp. 76-85.

David M. Nichols, Kirsten Thomson, and Stuart A. Yeates, 2001. "Usability and Open Source Software Development," In: E. Kemp, C. Phillips, Kinshuck, and J. Haynes (editors). *Proceedings of the Symposium on Computer Human Interaction.* Palmerston North, New Zealand: SIGCHI New Zealand, pp. 49-54.

Seth Nickell, 2001. "Why GNOME Hackers Should Care about Usability," GNOME Usability Project, at http://developer.gnome.org/projects/gup/articles/why_care/, accessed 28 November 2002.

Jacob Nielsen, 1993. *Usability Engineering.* Boston: Academic Press.

Donald A. Norman and Stephen W. Draper (editors), 1986. *User Centered System Design: New Perspectives on Human-Computer Interaction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

Tim O'Reilly, 1999. "Lessons from Open-Source Software Development," *Communications of the ACM,* volume 42, number 4, pp. 32-37.

Eric S. Raymond, 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, and at http://www.oreilly.com/catalog/opensources/book/raymond2.html, accessed 28 November 2002.

Eric S. Raymond, 1998. "The Cathedral and the Bazaar," *First Monday,* volume 3, number 3 (March), at http://firstmonday.org/issues/issue3_3/raymond/, accessed 28 November 2002.

Eric S. Raymond and Guy L. Steele, 1991. *The New Hacker's Dictionary.* Cambridge, Mass.: MIT Press.

Walt Scacchi, 2002. "Understanding the Requirements for Developing Open Source Software Systems," *IEE Proceedings — Software,* volume 148, number 1, pp. 24-39.

Jean Scholtz, 2001. "Adaptation of Traditional Usability Testing Methods for Remote Testing," *Proceedings of the 34th Annual Hawaii International Conference on System Sciences.* New York: IEEE Computer Society Press.

Jean Scholtz, 1999. "A Case Study: Developing a Remote, Rapid, and Automated Usability Testing Methodology for On-Line Books," *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences.* New York: IEEE Computer Society Press.

Ben Shneiderman, 2002. *Leonardo's Laptop: Human Needs and New Computing Technologies.* Cambridge, Mass.: MIT Press.

Elissa D. Smilowitz, Michael J. Darnell, and Alan E. Benson, 1994. "Are we Overlooking Some Usability Testing Methods? A Comparison of Lab, Beta, and Forum Tests," *Behaviour & Information Technology,* volume 13, numbers 1 & 2, pp. 183-190.

Suzanna Smith, Dave Engen, Andrea Mankoski, Nancy Frishberg, Nils Pedersen, and Calum Benson, 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 November 2002.

Bruce Sterling, 2002. "A Contrarian Position on Open Source." presented at the *O'Reilly Open Source Convention,* Sheraton San Diego Hotel and Marina (22-26 July), San Diego, Calif., at http://www.oreillynet.com/pub/a/network/2002/08/05/sterling.html, accessed 28 November 2002.

John C. Thomas and Wendy A. Kellogg, 1989. "Minimizing Ecological Gaps in Interface Design," *IEEE Software,* volume 6, number 1, pp. 77-86.

Matthew Thomas, 2002. "Why Free Software Usability Tends to Suck," at http://mpt.phrasewise.com/discuss/msgReader$173, accessed 28 November 2002.

Peter Thomas and Robert D. Macredie, 2002. "Introduction to the New Usability," *ACM Transactions on Computer-Human Interaction,* volume 9, number 2, pp. 69-73.

Jennifer A. Thompson and Robert C. Williges, 2000. "Web-based Collection of Critical Incidents during Remote Usability Evaluation," *Proceedings of the 14th Triennial Congress of the International Ergonomics Association and the 44th Annual Meeting of the Human Factors and Ergonomics Society (IEA 2000/HFES 2000),* Santa Monica: Human Factors and Ergonomics Society, volume 6, pp. 602-605.

Peter Trudelle, 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, Minn., at http://www.iol.ie/~calum/chi2002/peter_trudelle.txt, accessed 28 November 2002.

Tom Tullis, Stan Fleischman, Michelle McNulty, Carrie Cianchette, and Margaret Bergel, 2002. "An Empirical Comparison of Lab and Remote Usability Testing of Web Sites," presented at the *Usability Professionals Association Conference* (July), Orlando, Fla.

Hal R. Varian, 1993. "Economic Incentives in Software Design," *Computational Economics,* volume 6, numbers 3-4, pp. 201-217.

Marco A.A. Winckler, Carla M.D.S. Freitas, and José Valdeni de Lima, 1999. "Remote Usability Testing: A Case Study," In: J. Scott and B. Dalgarno (editors). *Proceedings of the Conference of the Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia (OzCHI'99),* Wagga Wagga, New South Wales, Australia, pp. 193-195.

Editorial history

Paper received 9 December 2002; accepted 24 December 2002.

Contents Index

The Usability of Open Source Software by David M. Nichols and Michael B. Twidale
First Monday, volume 8, number 1 (January 2003),
URL: http://firstmonday.org/issues/issue8_1/nichols/index.html