Journal of Interaction Science
a SpringerOpen Journal

RESEARCH Open Access

Open source and accessibility: advantages and limitations

Michael Heron[1*], Vicki L Hanson[2] and Ian Ricketts[2]

Abstract

In this paper we discuss the open source process as it relates to accessibility software. Open source is a development model that has shown considerable benefits in a number of application areas. However the nature of accessibility tools and the intended users of such software products raise issues that must be addressed by the developer before users encounter the tools in real world contexts. In this paper we discuss the nature of the open source process, how it functions, and the motivations with regards to participation that developers self-report. We then explain the impact of these elements of the open source process as they relate to adaptive accessibility software. We use some specific examples of issues raised from the adoption of open source via a discussion of the ACCESS Framework, an accessibility engine designed to provide cross-platform accessibility support through plug-ins.

Keywords: Open source, Human factors, Accessibility, Usability, Inclusivity, Gamification

Introduction

Running parallel to the more traditional models of commercial software development is a philosophy known as the **open source movement**. The movement is large and diverse, but at the core centres around a basic agreement that it is important for a software product to be distributed alongside the source code of its implementation. As a natural consequence of this, the open source movement is strongly involved in issues of copyright, redistribution and the rights of individuals to make derivate and transformative works from an original implementation. Open source is a common practise in computing science and many of the core technologies of the internet and desktop have been developed and distributed under one or more of the permissive licences generally accepted as conforming to the principles of the movement.

However, for all its importance in the development of computer artefacts, there is little literature available investigating the implications of providing open source software for inexperienced users. Open source software is not simply the same as commercial software that is provided for free - the process of open source contribution, whether it be software or knowledge, is an unusual and relatively novel approach to building information artefacts.

The particulars of the open source development model is such that it has numerous implications for those looking to deploy particular kinds of software for specific demographics. In this paper, we provide an overview of some of the implications for one specific situation – adaptive accessibility software. To frame this argument properly we must first discuss how open source as a development model functions and how the motivations of those who contribute influence the end product. We then highlight some of the implications of this process with regards to a specific open source accessibility tool developed by the authors.

Open source development

At its simplest, an open source application is one in which the source-code that builds the application is distributed alongside, or instead of, the executable binaries that comprise the program itself [1]. For more substantial definitions, there are several organisations that have claimed philosophical ownership over the term, such as the Open Source Initiative (OSI) who have layered several other principles onto the core definition [2]. Others such as the GNU project have made the term part of a larger movement within software development known as

* Correspondence: michael.heron@canterbury.ac.uk
[1]Canterbury Christ Church University, Canterbury, Kent CT1 1QU, United Kingdom
Full list of author information is available at the end of the article

the ‘free software movement’ within which the openness of the product becomes the key driving principle [3].

It is the nature of the terms under which a software application is distributed (the licensing agreement) that defines the freedom that users have with the software. Some licences prohibit modification for profit, while others act virally and require the adoption of the licence in all derivative works. Some permit their use in all circumstances, others have restrictions for those using the software for commercial purposes. The exact enumeration of developer and user rights is primarily a concern for those wishing to extend or modify the software rather than for those end users simply looking to take advantage of software that is freely downloadable.

The open source movement has attracted much ‘free’ labour to the cause of building open, transparent software systems. Despite the peculiarities of the approach, it has resulted in many substantial open source projects becoming reliable, scalable technologies that have been used at all levels of the digital economy. Open source software is used for everything from individual servers to the hardware that runs mission critical systems for multinational corporations. Some of the more significant examples of this kind of software include:

- technologies that underpin the infrastructure of the Internet such as Apache, Perl, PHP and MySQL;
- operating systems used for both servers and desktops such as Linux and Haiku;
- web browsers such as Firefox and Chrome;
- desktop application software such as OpenOffice, GIMP and Blender; and
- web applications such as Mediawiki, phpBB and Wordpress.

The collaborative movement has boasted from the start many successful products competing directly with closed-source, proprietary alternatives. Even so, some authors (for example, [1]) have raised concerns about the suitability of open source as a process for developing commercial software.

The largest and most successful of open source projects are community outcomes, resulting from the large-scale integration of the work of multiple contributors. The product of this contribution tends to follow the usual long-tail pattern whereby a small minority are responsible for the majority of content (such as is noted in [4,5]).

In open source projects, individuals choose elements of a common resource to implement or improve, and the modifications are then folded into the original, often with some kind of central authority acting as an intermediary for approval [6], p. 53). Aside from a few core individuals, contributors to such projects do not usually receive any kind of financial reward for their efforts. Instead, the context in which they operate more closely resembles that of a traditional gift culture ([6], p. 81; [7]) in which one’s personal standing is increased not by the amount an individual has, but by the amount an individual gives away. However, as a counterpoint to this, it should be noted that many of those who work most closely with the open source movement are remunerated as part of their job as advocates within larger technology organisations.

This collaborative approach to development has since extended beyond its roots in software source-code into areas such as collaborative knowledge creation, of which the most successful of these is the peer-produced knowledge resource known as Wikipedia. It is clear that collaborative production of resources has resulted in a great deal of concern for companies built on more traditional models of development (For some examples, see [6,8], p. 183; [9]).

Motivation to contribute

It is not immediately apparent why a process built on voluntary, unremunerated participation can result in serviceable products. Only a minority of individuals engaged in open source development are employed and financially compensated by the larger companies dedicated to open source initiatives [10]. Participants are giving their time and often marketable skill-sets to projects whereby any financial return (if any such return exists) accrues to other individuals or organizations. Several reasons for this have been proposed as a result of studies into motivation of open-source software development:

- Perfecting expertise ([6,11]; Lakhani & Wolf, [12]; [13])
- Enhancing reputation ([14]; Lakhani and Wolf, [12]; [6,13])
- Fun and enjoyment ([13]; Lakhani & Wolf, [12]; [15])
- Expectation of reciprocity [6]
- Job Market Signalling [6,16]
- Altruism [17]
- Belief in a principle of development [6,18]
- To fulfil a personal need [6,19]

All of these authors have stressed different motivations to different degrees. Most studies have treated the open-source phenomenon as a unified whole, but it does not hold that motivation will be identical across different domains of participation [20]. Oreg and Nov [21] discuss how the context of the collaboration impacts on the motivation of participants. These studies have focussed primarily on those who volunteer often specialised skill-sets to a project, and their results are not necessarily transferable to the more open participation of projects such as Wikipedia.

The expectation of generalized reciprocity also underlines much of the nature of open-source participation. The expectation is not that there is a mapping between contributor and beneficiary, but that at some point the contributor will benefit from the work of someone else in the community [16,22].

Open source participation

Having dealt with the issue of why people participate, the next issue to be addressed is how. The nature of open source development prohibits the use of a traditional, top-down management structure due to the exponentially increasing co-ordination and transaction costs [23]. Instead, open source projects are traditionally defined by three key principles:

- The ability of individuals to self-select their contributions ([6], p. 31-32, 57-58; [24])
- The ability of individuals to choose their level of participation ([25], p. 70)
- The ease at which individual contributions can be integrated into the whole ([6], p. 32; [11], p. 77-82).

The first principle allows for a high degree of efficiency in the allocation of individuals to tasks. In an open-source project, participants are not assigned a task to complete by the project leaders. Instead, they identify an area of the project in which they are interested, and in which they feel they have the competencies to make an improvement ([6], p. 31; [11]). This method ensures that no individual is given a task for which they have no competency, or are allocated a duty which they find personally onerous. Critics of the open source model claim that this leads to uneven distribution of attention across an application. It is argued that the high-profile, high-prestige tasks get disproportionate attention compared to the low-profile, low-prestige tasks such as application documentation. Additionally, there are concerns that while open-source as a method thrives in 'interesting' problem spaces, that it has issues producing applications of a more mundane nature [14].

The strength of this approach though comes in the heterogeneous makeup of the developer pool for open source applications. What seems to be an insurmountable task to one developer may be trivial to another simply because their skill-sets, experience and mental models are especially amenable to that scenario ([6], p. 32). This principle is summed up informally by Raymond [6], p. 3 as Linus' Law: 'Given enough eyeballs, all bugs are shallow'.

Coupled to this is the ability of an individual to choose their level of participation in the project. An individual may choose to develop entire new features, or focus on fixing small bugs in existing features as they desire. Allowing highly granular levels of participation ensures that individuals are never allocated tasks that exceed their level of motivation to contribute (Tapscott & Williams, [25], p. 70). An individual may have the skill-set and the mental model needed to implement a specific piece of code, but they also need the will to do so. In order for an open-source application to support this, it has to permit people to participate at all levels of the process, from finding bugs to fixing bugs to suggesting features to implementing features.

The nature of open-source development is meritorious in most cases – those central authorities that exist have risen to their positions largely as a result of their standing as initiators and sustainers in a project, and through demonstrated authorial credibility ([11], p. 78-84, 321; [6], p. 84-92; [26]). Individuals who have demonstrated the ability to leverage this credibility include Linus Torvalds, Jimmy Wales and Richard Stallman. Their ability to influence development and contribution persists as long as their credibility remains. Leadership is thus an on-going, emergent property of the process of development.

In cases where a leader loses this credibility, ownership of the project may be removed by 'forking' the code. A rival development team begins their own parallel development from a common code-base which becomes a 'fork' of the original with its own contributors and design philosophies. Such actions are not usually looked upon positively in the open source community ([6], p. 85; [11], p. 171) due to the way they fracture loyalties and contributors. The threat of a fork though serves a purpose as the ultimate moderator of authority and a mechanism for ensuring multiple development directions can potentially be explored.

Advantages and limitations of open source

Outside the context of research there are many compelling reasons for an author to release their software as open source – amongst these are the expectations that offering code for free will increase market share [27] and the belief that building a development base around the tool will increase long-term sustainability [28] as well as increase the personal reputation and future employability of the author [29]. For those not looking to commercialise their software products, open source offers a range of potential benefits.

Related to this is that many software projects are developed because the author has a need for them [6,30] or because the author seeks a particular creative outlet. These projects then get released to the larger development community and to end users. This increased pool of interested parties results in further successive improvements being made to the software. Each party may be acting in their own self-interest by contributing, but all benefit exponentially from the process.

Within the context of academia there are also philosophical issues such as adhering to the principles of open research. Providing full access to freely modifiable source code can be difficult when coupled with the restrictions associated with proprietary software. Additionally, publishing the algorithms and software diagrams may show the general shape of a solution, but there are implementation subtleties that make it difficult for others to recreate the software in a way that allows for replicating and corroborating of empirical results. These and related issues argue strongly in favour of the release of open source research code along with publications.

These advantages however come with a suite of limitations that can limit the usefulness of open source as a process when applied in certain environments.

End user documentation, by virtue of the fact that the audience of such material is not part of the 'developer culture' offers less opportunity for meeting the motivations of open source developers. It does little for enhancing personal reputation, offers few opportunities for perfecting expertise, gives little benefit for job signalling, and is rarely considered to be fun or enjoyable. Poor quality end-user documentation is not a specifically open source concern, but becomes especially critical when one considers that open source software is expected to evolve as time goes by. In all but the most rigorously managed open source projects, software is always in a draft state.

Related to this, the nature of authorial leadership as discussed above means that a 'philosophical vision' for where a project is going can result in significant changes being pushed version to version. It is relatively common for open source software to change dramatically between updates, often with little regard for how it impacts on common routines of established users. One good example of this comes from when the GNOME windows manager software was switched over, by default, to using a new and unfamiliar metaphor for exploring a desktop, as discussed in http://www.osnews.com/story/7548. One quote from that article in particular is of relevance:

"'That's why the Gnome team chose to use the spatial mode as the default: expert users who want file hierarchies can change that within 20 seconds. Novice users, on the other hand, should never need to see a file hierarchy. It just makes sense this way."

The constantly evolving nature of open source software means that documentation must be updated to reflect each change in the user experience. Developers often over-estimate the computer literacy of novice users [31], and as such even when the documentation is being well maintained it often fails to be presented in a way that is tractable for novices.

For academic research software in particular, long-term support is not something that is feasible to provide to the community. Even those research grants that stipulate an open source release of software rarely include budgeting for long term support and maintenance. The expectation is that the community will handle that – in some spheres, that is a reasonable expectation. For software aimed at novice users, that is unlikely to be optimal.

Open source software is provided 'as is' – technical support for the majority of open source projects is limited to what time the developer can spend addressing questions, and the collaborative help available through the use of online forums, wikis and discussion groups.

Documentation in the form of manuals may be extremely limited, and erratically updated [32] as outlined above. Instead, support is most often provided informally as part of an online community. Such support can often be highly critical and combative, as is evidenced in the following extract from an arbitrary web-support forum[1]:

Seriously people.

If YOU cannot be bothered to do the simple and obvious task of Reading FAQs, SEARCHING for and reading related topics do... then do NOT expect US to answer you!

If YOU cannot fathom that you are meant to supply details when you aska question - incuding the basics such as URLs... then do NOT expect US to bother attempting to help

(http://www.google.com/support/forum/p/Webmasters/thread?tid=029f6d5729b35a89&hl)

Such posts are not uncommon, and while they reflect a perhaps understandable general frustration with those who display a lack of concern or understanding of netiquette, novice users run the danger of being on the receiving end of disproportionately aggressive and personal criticism for doing nothing worse than asking a question on a forum specifically intended to ask for help.

Additionally, the language used in many forums can be problematic in its technicalities. One of the older users in one of our studies wrote: '*I find online help useless as it's written in American/technical language which, though using English words, makes no sense whatsoever*'. There can be a considerable mismatch between the language used as a matter of course within an open source forum, and the language used by those seeking assistance.

Those applications that are well supported are usually done so on a service model where the product itself is free, but the technical support is billed. For a user unfamiliar with new technology, this can be prohibitively expensive unless subsidized by an employer or other organisation. Coupled to this is the fact that open source software often suffers from usability issues [33,34], and this creates a pair of linked problems that are likely to cause substantial complications in providing software support to non-technical users.

Certain kinds of technical architectures lend themselves more easily to developing a community of open source documentation support than others. A simple forum with a search box will be sufficient for many projects, but when users may not know what it is they don't know a more structured approach is required. Centralised indexing requires time and effort that may not be available, so providing community indexing facilities such as tagging and rating can help bridge the gap. Inculcating the virtues of the open source movement itself into the community can likewise help. Building a sense of the value of reciprocity means that those who work out their own problems are likely to report back to the community the problem they had and the solution they discovered. This would aid in alleviating the frustration of the situation described in http://xkcd.com/979/.

While searching for information online is unlikely to be a problem for most demographics, there are others that are less likely to seek out such information in the first place, more wary of the information that they find, and more distrustful of software generally. Steps must be taken to build credibility and sustain that credibility whilst also ensuring that the act of seeking help is not rewarded with aggression.

Finally, open source software by virtue of its underlying philosophy introduces security considerations – a different profile of security is required. The concept of 'security through obscurity', while flawed as a single measure, is commonly employed in a system known as 'defence in depth' where multiple layers of countermeasures exist to flummox attempts to violate the integrity of software or data. The impact of open source on security remains a controversial topic [35], but the risks associated with exposing end users to security risks in accessibility and universal access domains [36] must be assessed, analysed and addressed.

Accessibility software

A particular category of software employed by users with physical or cognitive impairments is known as **accessibility software**. These are tools aimed at changing the way in which a user works with a system with the intention of reducing barriers to interaction. Issues that are known to create interaction difficulties include blindness or partial sight, colour blindness, deafness, and the physical inability to work with input devices such as keyboards and mice. In some cases, a distinct impairment may be possible to identify and correct for. In others, a more subtle interaction of minor ailments may make it difficult to identify a single causative factor and subsequently complicate the search for an appropriate solution.

Most operating systems come complete with a suite of accessibility tools that are primarily toggles or sliders that can be set to improve the user experience for those with conditions that can be easily identified. These range from magnifiers that can be used by those with sight impairments, to adjustments to the speed and accuracy of the mouse. These accessibility tools are 'fire and forget' and do not require long term monitoring or analysis of user input.

There exists however a category of accessibility software that seeks to perform on-going adaptation of a user's environment based on an analysis of their input via various methods. Previous work aimed at addressing issues with keyboard interactions includes Koester, Lopresti and Simpson [37]; Trewin [38]; Trewin [39]; Heron, Hanson & Ricketts [40]. Adaption to mouse interactions has been discussed in Trewin, Keates and Moffat [41]; Wobbrock, Fogarty, Liu; Kumuro and Harada [42]; Heron et al [40]. These tools seek to provide regular adjustments to a user's context with the intention of correcting issues as they are algorithmically detected.

With the former category of software, there are few significant risks posed by open source as a development strategy that are not common to all systems. For the latter, the nature of the open source process itself introduces a number of elements that must be addressed and dealt with. Adaptive correction requires a relatively deep analysis of user interaction, which in turn requires the software to be able to hook into streams of input from user devices – keyboard and mouse primarily, but research tools in specialised contexts may incorporate less standard streams (See [36] for a discussion of some of these). An accessibility tool aimed at improving keyboard interactions needs to read each of the keys that a user enters regardless of the application within which they may be working. In short, a keyboard accessibility tool must function as a sophisticated key logger. It is possible to scramble key input before it gets to the tool, but in doing so certain kinds of accessibility adaptations become impossible to implement. Trewin [38] would be possible with scrambled keyboard input, but Trewin [39] by comparison would not.

Accessibility tools may also require access to contextual information about the user's environment. They may need to know within what application a user is working. A tool designed to identify double click errors for example is unlikely to be able to do so accurately if it cannot tell the difference between a word processor and a computer game. The accessibility issues within computer games (as discussed in [43]) will require their own specific adaptations that cannot be easily generalised to the rest of the system.

To be able to perform a full range of adaptations, software must have access to unmodified input streams from any relevant input device; must be able to track user context; and usually must store historical data of user interactions even if only for a short time. The only feasible way in which such adaptive software can make meaningful corrections is to draw in these different sources of information and aggregate them into some form that can be analysed.

The installation of such software then requires either a controlled environment kept separate from a user's 'real life', or a tremendous amount of trust on the part of a user. Developers of such accessibility software then must operate under a significant duty of care burden and ensure that the development route that they go through does not significantly negatively impact on the future user-base of the tool. These are primarily issues that a developer must consider rather than something that would inform the decision making of the majority of end-users.

Issues of accessiblity: the ACCESS framework

To illustrate why open source must be carefully evaluated in the development of accessibility software, a case study of an adaptive accessibility tool will be provided. This tool was developed internally within the University of Dundee and then released as an open source project (available at https://github.com/drakkos/ACCESS). The nature of the tool highlights several of the issues associated with open source adaptive accessibility tools.

The tool itself is called ACCESS (Accessibility and Cognitive Capability Evaluation Support System) and is designed as a software framework that provides adaptive accessibility support across a user's computer. ACCESS itself contains no usability corrections. It instead delegates the responsibility for identifying and then correcting interaction issues to the plug-ins that can be incorporated at run-time. The framework works across multiple operating systems and provides a common API for plug-ins to interact with the deployment platform regardless of the environment in which they were written. The technical specifics of this are irrelevant to the case study, but are fully discussed in Heron [44].

The framework reads all keyboard and mouse input from the user, and keeps track of the applications within which they are working. It allows for plug-ins to make changes to the system settings of the underlying operating system, and similarly allows them to read in the current state of system settings. It has routines for executing external applications, reading input from invoked commands, and provides a socket architecture that allows for plug-ins to communicate with external applications.

All of these facilities are used by the plug-ins that have been developed for the Framework – examples of these include a plug-in that dynamically adjusts the double-click threshold in response to user double-clicking difficulties; a plug-in that records all user input and allows it to be played back for analysis and testing; a context sensitive 'post it note' system that changes its contents depending on what application is currently active, and several others. The range of support tools that can be developed is considerable, and it is the access to the user's interaction patterns that makes this breadth of functionality possible.

Currently, all plug-ins were developed internally by the primary author of this paper, and were deployed only in research scenarios for evaluation [40]. However, the design of this tool introduces numerous complexities for making it available on a wider basis, and these stem from the implications of releasing it under an open source licence.

User support and the ACCESS framework

As discussed above, some critics of the open source model believe that it thrives most in interesting problem spaces – documentation is rarely considered to be one of these, and the end-user documentation issues that currently exist in ACCESS demonstrate that this is the case. As a research tool developed within a particular funding context, the only author time available for further documentation must come at the cost of pursuing new research or future possibilities for expansion. Documentation in its usual form is unlikely to yield any research outputs, and this can greatly influence the extent to which end-user document can be written, polished and updated as the system is changing.

As ACCESS evolves, new capabilities will be introduced, new plug-ins will be developed, and each of those plug-ins will likely have their own interaction subtleties that other researchers may not document. User documentation that is available now will require constant updating to reflect changes made, and the time to do that documentation is not part of the research funding. As discussed above, this is likely a task for which individuals will not regularly self-select. However, there exist possibilities by which individuals can be encouraged to self-select their contributions in the ways that benefit the project most. The growing interest in the subject of gamification (As defined in [45]) offers intriguing future possibilities for the highly visual rewarding of those who contribute to parts of the system that are not otherwise seen as high prestige. The awarding of prominent badges for contribution, in the style of game achievements [46] could potentially drive contributor attention to otherwise ill served elements of the system. Future plans for ACCESS will look to incorporate such a framework for contributor recognition in the hope that it has a positive impact on the diversity of areas to which developer attention is directed.

Most open source projects, certainly those that begin life as research software, cannot realistically provide a coherent strategy for user documentation and long term support. Instead, it has to be something that is based primarily on hope – that those collaborating will go against the usual norms and produce the necessary documentation and supportive community required. These communities in turn must be heavily moderated and aggressively policed – deputizing trusted 'elder members' of the user-base in this respect can go a long way towards creating the right kind of environment where those with queries are not sent away unhappy.

Within ACCESS, we cannot expect that the target users will be comfortable with 'fiddling around until it works', and as new operating systems come into use the framework will need to be updated to reflect the changing context in which users function. Considering the demographic groups for which ACCESS is designed, this is an important issue that requires careful management for when the tool is eventually delivered to end-users.

Open source and the plug-in architecture

As discussed above, ACCESS delegates the responsibility for identifying usability issues to plug-ins. This means that each plug-in has access to a huge amount of sensitive information. Opening up the framework to an open source development model means that new and innovative plug-ins can be developed by interested parties. It also means that the feature set of the framework itself can be modified and expanded to support the needs of researchers and other stakeholders in accessibility software. However, the power inherent in a plug-in creates a very real danger to users. It would be trivial in the framework discussed in this paper to develop a context sensitive key and mouse logger, pulling in likely passwords when the user was within a web browser and then sending those passwords off to a remote location. Plug-ins can be developed openly and then distributed through the Internet, meaning that they could realistically end up being a significantly dangerous vector of attack for malware. Many of these issues are further discussed in Bahr et al. [36].

The plug-in model is effectively employed in many open source software packages, where extra or enhanced functionality is available for download. The Firefox browser is one notable example, as is the Netbeans development environment. In many ways, the development of self-contained apps for various smartphone ecosystems can also be considered a system of distributed plug-ins aimed at enhancing the functionality of a platform both separately and as an integrative element in a larger, more holistic experience.

For ACCESS and its intended user base, there is too much risk is simply allowing anyone to install any plug-in they find on the Internet beacause a substantial amount of potentially sensitive interaction data is being drawn into the plugins. Data pulled into plugins can include usernames, passwords and credit card numbers. Allowing any plugin to read input indiscriminately and manipulate it without restriction represents a significant security risk within a plugin obtained from an unscrupulous source. If a framework is open enough, there is no way a technically unsophisticated user can tell what any given plug-in is doing even if the source code is freely available. Some method of ensuring the trustworthiness of plug-ins is required to ensure that advantage is not taken of inexperienced members of the user-base.

Ceding a little development efficiency in exchange for a centralised repository of 'trusted plug-ins' is an effective mechanism for limiting the possibility of damage. There are other technical solutions, such as running plug-ins in a sandbox. A sandbox is a security measure used to limit the access of a piece of software to certain sensitive parts of a running system. This is inappropriate with regards to adaptive accessibility software which will, as a matter of course, need very low level access to the operating system in order to perform evaluative or corrective actions although in other domain spaces such as mobile apps it is more appropriate [47].

Digital signing of plug-ins, ensuring that new plugins are made available only once they have been approved by some central authority offer a good deal of flexibility in resolving these issues. This is only viable as a solution as long as users are installing the 'official' version of the core engine. A discussion of the issues relating to the dangers of violating the integrity of a trusted platform can be found in Felt, Finifter, Chin, Hanna, Wagner [48].

If the core engine of an application is open-source, there is little to stop an individual making their own modified version of the engine and distributing it under a different (or even the same) name. Individuals are always free to fork an open source project if so desired, meaning control of the source-code can never be a guaranteed right. It cannot be assumed that the core engine installed by a user is the 'official' version. A rogue version of the core engine could use a different source to authenticate the signing of plug-ins, or remove the signing entirely. Luckily, there is some evidence to suggest that the typical user-base for such tools tends to be more wary of software from unknown organisations, and that if an application purports to be associated with a known brand that they will investigate the veracity of that claim. 61.18% of respondents to a questionnaire distributed as part of this study (N = 296, M = 121, F = 175, average age = 68.8) indicated that they were wary of installing new software that came from unknown publishers, and 53.95% indicated that they felt a brand was important in building confidence in new software. 49.34% indicated that they would investigate the claims of an application to be associated with a particular brand. Software available for free was trusted less amongst this group than commercial software – 9.54% agreed they would trust a piece of commercial software from an organisation they did not recognise, compared to only 6.91% who would trust free software from an organisation they did not recognise, where $t(266) = 4.45$, $p < .000$.

These are results that are consistent with focus group comments made during the testing of the ACCESS framework discussed above. ACCESS itself was developed through the IBM Open Collaboration Project and as part of the accessibility research within the school of Computing at Dundee University. These are brands that can be used to inspire confidence in end-users. Ensuring that such a tool is associated with a well-known and trusted brand can potentially go a long way to ensuring that rogue core engines are not installed, but it cannot eliminate the risk entirely.

It is important too that all opportunities are taken to raise awareness of the dangers of installing software from untrusted sources. Modern versions of the installation routine for the BitTorrent application contain the following warning:

> *A number of websites have created their own versions of the BitTorrent client and plug-ins that attempt to charge money for the applications and infect your computer with malicious code. To protect yourself, be sure to only download our software from www.bittorrent.com or one of our authorized distribution partners: www.download.com, www.sourceforge.net, www.tucows.com, and www.jumbo.com.*

If such software is downloaded in good faith from a disreputable site, then a user's computer can be entirely compromised with no easily identified outward signs of infection (for some examples, http://news.bbc.co.uk/1/hi/technology/7907635.stm, http://www.eweekeurope.co.uk/news/web-users-prone-to-fake-av-2-13384). Users can be scammed into believing that they are liable for criminal damages (as in http://torrentfreak.com/malware-extort-cash-from-bittorrent-users-100411/), or strong-armed into buying software to fix purported malware infections.

The solution that we are exploring for ACCESS incorporates a strong degree of control over authorisation of new features, and also on distribution of the application itself. Most licences stipulate that any user is permitted to modify and distribute the software provided there is appropriate attribution. An open source licence of this type cannot be ethically used to distribute adaptive accessibility software if in doing so it potentially opens up its intended user-base to danger. It cannot be assumed that disreputable individuals will honour the provisions of a licence in any case, but the onus is on the developer to ensure the licence they pick does not **enable** abuse. At best licences stop ethical developers making changes, and provide some mechanisms by which abuses can be policed.

One approach that has been shown to address this issue is to distribute the code framework under one name, and trademark a brand name for the operational version, as was done for Google's Chrome web-browser. The code for Chrome is freely available, but Google circumvents the problem of multiple versions of their application being released by placing the brand name under trademark protection. This gives a legal recourse in the event of rogue developers appropriating the code. The financial burden that is implied by aggressively protecting such a trademark puts it beyond the resources of most hobbyist developers and externally funded researchers. It is also dependant on the trademark name being the one that gains mass traction in the minds of users – Google can easily ensure this by leveraging their tremendous Internet presence. Smaller organisations or individuals usually do not have such resources available.

There are unfortunately no universally applicable rules that allow for the developer of an open source accessibility tool to prevent abuse. Awareness of the issues and designing software with the potential consequences in mind allows for the risks to the end-user to be minimised. Specifically with relation to ACCESS, an analysis of the risks allowed us to make the following design decisions which have influenced future plans for the deployment of the tool to end users:

1. Associate the software with a trusted brand if possible, such as a university, charity, or software publisher.
2. Ensure a single, certifiable location from which the 'official' installer can be downloaded.
3. Incorporate digital signing of plug-ins so as to ensure they are reviewed before being made available to end users.
4. Make available the necessary technical architecture to support genuinely collaborative community support.
5. Inculcate a good 'civic'mind-set for support by recruiting the most helpful of contributors as moderators.
6. Provide public recognition for contributions that are considered 'high value' for the project in the form of visible achievements.

While these design decisions do not eliminate the risks of open source distribution they do help manage it whilst still allowing for the benefits associated with open source development to be effectively harnessed and directed in the right kinds of ways to the right kinds of areas.

Conclusion

The nature of adaptive accessibility tools means that much sensitive data must be made available to those that develop them. When working within a closed-source model under the auspices of a known and trusted organisation, the managerial hierarchies and legal responsibilities can serve to ensure that abuses are minimised if not necessarily entirely eliminated. Within a for-profit company, ongoing support can be justified as a legitimate business expense, and good documentation can reduce the cost of supporting software that is released.

The open source model however does not permit for easy solutions to issues of providing long term support or comprehensive user documentation. The need to open up the source code and to allow for the distributed development and then integration of systems allows for a highly efficient development model. In the area of adaptive accessibility tools this creates dangers if the process is not effectively managed or the risks not fully understood. The highly sensitive nature of data routinely analysed by adaptive accessibility tools requires that developers consider the duty of care that goes along with releasing such tools into the wider community. Brand association, conscious shaping of community norms and management and moderation of plug-ins can go some way towards managing the risks, but they cannot be entirely eliminated.

Some of the factors that encourage individiuals to contribute to open source projects have their basis in reputation, recognition, and signalling for job markets. Appealing to these desires via achievements and other gamification techniques could potentially serve as a way by which some of the inherent problems of self-selection could be effectively addressed. Qualitative and quantitative results within this context are not currently available by which their effectiveness could be assessed, but these form one of the routes available for those looking to direct attention towards specific parts of their open source projects.

Endnotes

[1] This web forum is not aimed at novices in particular.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors were involved in the planning, design, conduct, writing and reviewing of this paper. All authors read and approved the final manuscript.

Acknowledgements
The research described in this paper was conducted as part of an IBM Open Collaborative Research project between IBM Research and the University of Dundee. Additional funding was provided by a Royal Society Wolfson Merit Award to the second author and by grant RCUK EP/G066019/1 "RCUK Hub: Inclusion in the Digital Economy".
Special acknowledgements go to Dr. Norman Alm, Professor Peter Gregor, Professor Alan Newell, and Dr. Janet Hughes, all of the University of Dundee, for their suggestions, advice and guidance. Thanks go to Dr. Shari Trewin of IBM and Pauline Belford for their very helpful feedback and suggestions.

Author details
[1]Canterbury Christ Church University, Canterbury, Kent CT1 1QU, United Kingdom. [2]University of Dundee, Dundee City DD1 4HN, United Kingdom.

Received: 5 March 2013 Accepted: 5 March 2013
Published: 7 May 2013

References
1. Gurbani, VK, Garvert, A, & Herbsleb, JD. (2005). *A case study of open source tools and practices in a commercial setting* (5-WOSSE: proceedings of the fifth workshop on open source software engineering volume 30th ed., pp. 1–6). New York, NY, USA: ACM.
2. Kirschner, B. (2008). *Building a balanced scorecard for open source policy and strategy: a case study of the Microsoft experience* (ICEGOV '08: proceedings of the 2nd international conference on theory and practice of electronic governance, pp. 226–231). New York, NY, USA: ACM.
3. Chopra, S, & Dexter, S. (2007). Free software and the political philosophy of the cyborg world. *SIGCAS Comput. Soc., 37*(2), 41–52.
4. Singh, V, & Twidale, MB. (2008). *The confusion of crowds: non-dyadic help interactions* (Proceedings of the 2008 ACM conference on Computer supported cooperative work, CSCW '08, pp. 699–702). New York, NY, USA: ACM.
5. Sowe, S, Stamelos, I, & Angelis, L. (2008). Understanding knowledge sharing activities in free/open source software projects: an empirical study. *Journal of Systems and Software, 81*(3), 431–446.
6. Raymond, ES. (2001). *The cathedral & the bazaar: musings on Linux and open source by an accidental revolutionary* (revisedth ed.). Sebastopol, CA, USA: O'Reilly and Associates.
7. Wu, CG, Gerlach, JH, & Young, CE. (2007). An empirical analysis of open source software developers' motivations and continuance intentions. *Information Management, 44*(3), 253–262.
8. Harmon, A, & Markoff, J. (1998). *Microsoft on trial: the new threat; an internal memo shows microsoft executives' concern over free software*. http://www.nytimes.com/1998/11/03/business/microsoft-trial-new-threat-internal-memo-shows-microsoft-executives-concern-over.html.
9. Giles, J. (2005). Internet encyclopaedias go head to head. *Nature, 438*(7070), 900–901.
10. Feller, J, Fitzgerald, B, Hissam, SA, & Lakhani, KR. (2005). *Perspectives on free and open source software* (pp. 3–22). Cambridge, MA: MIT Press.
11. Moody, G. (2002). *Rebel Code: Linux and the Open Source Revolution* (newth ed.). Cambridge MA, USA: Perseus Books.
12. Lakhani, KR, & Wolf, RG. (2005). *Why hackers do what they do: understanding 927 motivation and effort in free/open source software projects* (Social Science 928 Research Network Working Paper Series).
13. Von Krogh, G. (2003). Open-source software development. *MIT Sloan Management Review, 44*(3), 14–18.
14. Bezroukov, N. (1999). Open source software development as a special type of academic research (critique of vulgar Raymondism). *First Monday, 4*(10).
15. Moglen, E. (1999). Anarchism triumphant: free software and the death of copyright. Centro di studi e ricerche di diritto comparato et straniero. Available online at http://moglen.law.columbia.edu/publications/anarchism.html. Journal name: First Monday.
16. Hars, A, & Ou, S. (2001). *Working for free? - motivations of participating in open source projects* (HICSS '01: proceedings of the 34th annual Hawaii international conference on system sciences (HICSS-34)-volume 7). Washington, DC, USA: IEEE Computer Society.
17. Benkler, Y, & Nissenbaum, H. (2006). Commons-based peer production and virtue. *Journal of Political Philosophy, 14*(4), 394–419.
18. Hertel, G, Niedner, S, & Herrmann, S. (2003). Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy, 32*(7), 1159–1177.
19. Hippel, E. (2001). Innovation by user communities: learning from open-source software. *MIT Sloan Management Review, 42*(4), 82–86.
20. Moore, TD, & Serva, MA. (2007). *Understanding member motivation for contributing to different types of virtual communities: a proposed framework* (SIGMIS-CPR '07: proceedings of the 2007 ACM SIGMIS CPR conference on computer personnel doctoral consortium and research conference, pp. 153–158). ACM: New York, NY, USA.
21. Oreg, S, & Nov, O. (2008). *Exploring motivations for contributing to open source initiatives: the roles of contribution context and personal values. Computers in human behavior*. Amsterdam, The Netherlands: Elsevier Science Publishers.

22. Wasko, M, & Faraj, S. (2000). " it is what one does": why people participate and help others in electronic communities of practice. *The Journal of Strategic Information Systems, 9*(2–3), 155–173.
23. Benkler, Y. (2002). Coase's penguin, or, linux and the nature of the firm. *The Yale Law Journal, 112*(2), 369–446.
24. Anthony, D, Smith, S, & Williamson, T. (2005). *Explaining quality in internet collective goods: Zealots and good samaritans in the case of wikipedia*. Electronically. http://web.mit.edu/iandeseminar/Papers/Fall2005/anthony.pdf.
25. Tapscott, D, & Williams, AD. (2006). *Wikinomics: how mass collaboration changes everything*. New York: Portfolio Hardcover.
26. Reagle, J. (2007). *Do as i do: authorial leadership in wikipedia* (WikiSym '07: Proceedings of the 2007 international symposium on Wikis, pp. 143–156). New York, NY, USA: ACM.
27. Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly, 30*(3), 587–598.
28. Nyman, L, Mikkonen, T, Lindman, J, & Fougère, M. (2011). *Forking: the invisible hand of sustainability in open source software. In proceedings of SOS 2011: towards sustainable open source*. Tampere, Finland: Tampere University of Technology.
29. Hann, IH, Roberts, J, Slaughter, S, & Fielding, R. (2002). *First Evidence of Economic Incentives, Proc. 2nd Workshop on Open Source Software Engineering*. Pittsburgh USA: Carnegie Melon University.
30. Bonaccorsi, A, & Rossi, C. (2003). Why open source software can succeed. *Research Policy, 32*(7), 1243–1258.
31. Knight, J, & Jefsioutine, M. (2002). *Relating usability to design practice* (Proceedings of the 1st European UPA conference on European usability professionals association conference - volume 3, pp. 2–12). Swinton, UK, UK: British Computer Society.
32. Levesque, M. (2004). Fundamental issues with open source software development. *First Monday, 9*(4).
33. Nichols, DM, & Twidale, MB. (2002). Usability and open source software. *First Monday, 8*.
34. Raymond, ES. (2006). *The luxury of ignorance: an open source horror story*. http://www.catb.org/esr/writings/cups-horror.html.
35. Schryen, G. (2011). Is open source security a myth? *Communications of the ACM, 54*(5), 130–140.
36. Bahr, ML, & Gacey, H. (2011). Cyber risks to secure and private universal access. In C Stephanidis (Ed.), *Universal access in human-computer interaction. Design for all and eInclusion, volume 6765 of lecture notes in computer science* (pp. 433–442). Berlin Heidelberg: Springer.
37. Koester, HH, Lopresti, E, & Simpson, RC. (2007). Toward automatic adjustment of keyboard settings for people with physical impairments. *Disability and Rehabilitation. Assistive Technology, 2*(5), 261–274.
38. Trewin, S. (2002). *An invisible keyguard* (Assets '02: proceedings of the fifth international ACM conference on assistive technologies, pp. 143–149). New York, NY, USA: ACM.
39. Trewin, S. (2004). *Automating accessibility: the dynamic keyboard* (Assets '04: proceedings of the 6th international ACM SIGACCESS conference on computers and accessibility, pp. 71–78). New York, NY, USA: ACM.
40. Heron, MJ, Hanson, VL, & Ricketts, I. (2013). Accessibility support for older adults with the ACCESS framework. *International Journal of Human Computer Interaction*. doi:10.1080/10447318.2013.768139.
41. Trewin, S, Keates, S, & Moffatt, K. (2006). *Developing steady clicks: a method of cursor assistance for people with motor impairments. Assets '06: proceedings of the 8th international ACM SIGACCESS conference on computers and accessibility* (pp. 26–33). New York, USA: ACM.
42. Wobbrock, JO, Fogarty, J, Liu, SYS, Kimuro, S, & Harada, S. (2009). *The angle mouse: target-agnostic dynamic gain adjustment based on angular deviation* (Proceedings of the 27th international conference on Human factors in computing systems, CHI '09, pp. 1401–1410). New York, NY, USA: ACM.
43. Heron, MJ. (2012). Inaccessible through oversight: the need for inclusive game design. *Computer Games Journal, 1*(1), 29–38.
44. Heron, MJ. (2011). , The ACCESS Framework: reinforcement learning for accessibility and cognitive support for older adults. Dundee, Scotland: PhD thesis, Dundee University.
45. Deterding, S, Dixon, D, Khaled, R, & Nacke, L. (2011). *From game design elements to gamefulness: defining "gamification"* (Proceedings of the 15th international academic MindTrek conference: envisioning future media environments, MindTrek '11, pp. 9–15). New York, NY, USA: ACM.
46. Hamari, J, & Eranti, V. (2011). *Framework for designing and evaluating game achievements* (Proc. DiGRA 2011: think design play, DiGRA).
47. Miller, C. (2011). Mobile attacks and defense. *IEEE Security and Privacy, 9*(4), 68–70.
48. Felt, AP, Finifter, M, Chin, E, Hanna, S, & Wagner, D. (2011). *A survey of mobile malware in the wild* (Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM '11, pp. 3–14). New York, NY, USA: ACM.

doi:10.1186/2194-0827-1-2
Cite this article as: Heron *et al.*: Open source and accessibility: advantages and limitations. *Journal of Interaction Science* 2013 **1**:2.