Designing for Discovery: Opening the Hood for Open-Source End User Tinkering

Gifford Cheung
The Information School
DUB Group
University of Washington
Seattle, WA 98195 USA
giffordc@u.washington.edu

Braden Pellett
The Information School
DUB Group
University of Washington
Seattle, WA 98195 USA
bhp@u.washington.edu

Parmit Chilana
The Information School
DUB Group
University of Washington
Seattle, WA 98195 USA
pchilana@u.washington.edu

Shaun Kane
The Information School
DUB Group
University of Washington
Seattle, WA 98195 USA
skane@u.washington.edu

Abstract

According to the Free Software Movement, the user ought to have "the freedoms to make changes, and to publish improved versions" and "to study how the program works, and adapt it to your needs" [7]. The Open Source Initiative expects users to access source code, explaining that "you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy" [8]. These philosophies can shape a unique perspective on software usability that has not been addressed thoroughly in the open-source domain. That is: how to design user-interfaces and tools that facilitate access to source code and encourage the behaviors envisioned above, namely, to improve the code, to personalize it, to learn from it, and to share it. And, as the Open Source Initiative recommends, to make this easy. In addition to presenting this research perspective, we suggest some fruitful approaches to answering these questions and our current and future steps.

Keywords

Open Source Software, usability, user-centered design, design rationale, End-user Programming

ACM Classification Keywords
H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces---Evaluation/methodology, Theory and methods, User-centered design.

Introduction
Adam is brainstorming a new interface for the next version of his open-source web application, a content-management suite. His application is fairly popular, but he cannot keep up with the many incoming feature requests, some of which are too narrow to be worth his time and effort. He has a similar problem with bug-fixes and is looking for new ways to encourage more people to get involved in his project. He also notices that he has a new crop of users – website builders with little to no programming experience, many of whom are trying to bend his software to fit their agendas. So much activity and feedback are encouraging. This kind of energy is exactly why Adam made his software open-source in the first place, but now that the demand is there, is there anything more he can do to meet the needs of his project and these open-source users?

Three Challenges for Open-source Software
This paper is motivated by our desire to address a few challenges in open-source development. We've entitled these challenges as code reliability, customizability, and novice-programmer userbase. Our main goal is to address these challenges from an HCI perspective that is distinctly "open-source" in nature.

Code Reliability
Advocates of open-source code applaud it for being more reliable than closed source because "given enough eyeballs, all bugs are shallow"[1]. However, many open-source projects do not feature the support of a large group of contributors that make software like the Apache project reliable. While reporting on the Apache project, Mockus et al. [4] warn that "Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects in the released code." All open-source projects struggle with code reliability. If Mockus et al. are right, smaller or younger projects risk having defects in the code that may never be found or addressed. Thus, code reliability is a major challenge for open-source projects.

Customizability
Another challenge that motivates this paper is the challenge of customizability. If a project is open-source, one might expect that, if a user wanted to customize the code, he or she could modify it to meet a desired goal. We ask how a software project can make customization easier by using strategies *beyond releasing the source code publicly.* We believe that leaving the source-code open leads to promising challenges for the design of open-source software. Our perspective is that opening up the source code is only the first step for making software more customizable. The challenge then is to ask how HCI principles can work hand-in-hand with open-source to promote customizability.

[1] The Cathedral and the Bazaar. http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/

Novice-Programmer Userbases

Open-source software depends on the availability of its source-code to allow users to debug, customize, and extend it: presumably to free its users to do what they want with it. However, for users of open-source software who are novice programmers, source code can be as impenetrable as a binary executable[2].

One might ask if such users exist: people who are motivated to delve into the source code of a program despite having little to no programming experience. One example of this is in the field of bioinformatics. Within the last few years, open source tools have proliferated in the field of bioinformatics and, in fact, much of the scientific work in this domain is highly dependent on the availability of open source data sets and tools for manipulation and analysis. *BioPerl*, *BioJava*, and *BioPython* are examples of open source projects with a wide international user base of biologists and computer scientists. They offer libraries of modules and routines that can be used to connect bioinformatics applications and datasets for rapid development of an application. Within such scientific communities, users need to exercise precise control over these open-source tools but do not have the time to become expert programmers. Their focus is on the science, not the tool, and rather than spend time with a complex software tool, they sometimes abandon the software and resort to manual work instead. In this domain, complaints about open-source software include inconsistent method names, unclear interfaces, difficult documentation, and high requirement of learning a new programming language. These users recognize the potential power of open-source tools. They need lower barriers to customize and use the tools for themselves. The challenge is in designing software that fulfills its open-source promises for motivated, but inexperienced programmers.

[2] Fitzgerald and Agerfalk [2] suggest a similar critique by asking, “Is 10 million lines of code all that different from a binary executable?” The population that they refer to is broader than ours. It includes competent programmers who do not have the time or motivation to navigate an extremely large codebase. While an impenetrable codebase certainly is a roadblock for open-source software, we do not list it as a challenge because it is sufficiently addressed in the novice-programmer challenge and is not quite as high-level as the challenges for code reliability or customization.

An Open-Source Paradigm of Software Use

We have chosen these three challenges for open-source software as a way to highlight challenges that are especially relevant to open-source software. We see open-source software not merely as a policy that requires public availability of source code, but because we see it as representative of a ***paradigm of software use***. We derive this perspective from definitions that the Free Software Movement and the Open Source Initiative lay out for their software. The Free Software Movement’s definition [7] of free software refers to the following freedoms quoted below:

Freedom 0: The freedom to run the program, for any purpose; **Freedom 1:** The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this; **Freedom 2:** The freedom to redistribute copies so you can help your neighbor; **Freedom 3:** The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this.

They are similar to the Open Source Initative's definition for open-source software [8] which, for purposes of space, we do not include. Given by major organizations in the OSS movement, these definitions relate the *expected use* of the software by its users. Drawing from these texts, we have identified use cases meant to be common across all open-source software.

An Open-source Paradigm of Software Use

Use case 1: **Users can learn how the software works from the software**
Users are expected to study the program. The code must not be "deliberately obsfuscated".

Use case 2: **Users can improve the software**
Open-source endorses a vision of software that evolves and improves as it is used and reprogrammed.

Use case 3: **Users can personalize the software**
Open-source philosophy values the free use of software, "experimental modifications", and the right for users to do what they wish with software.

Use case 4: **Users can share the software**
Open-source philosophy values the free exchange of code to encourage growth & evolution of projects.

We have omitted use cases that are not emphasized by the open-source community as a whole (e.g. OSI Def 4. Integrity of the Author's Source Code). Also, we have omitted use cases which do not seem useful for deriving design recommendations for the software artifact (e.g. "Users can run the program for any purpose", FSM #0). Finally, we recognize that there is room for debate about the open-source vision for software use. Further discussion and debate is certainly welcome – this paper's primary contribution is to promote the view that there ***exists*** a paradigm of use for open-source software and that usability research can be directed by this paradigm.

A Design Approach to support the Open-source Paradigm

We see a correlation between these use cases and the challenges that we first introduced in this paper. The use cases are *solutions* to the challenges. Traditionally, these use cases have employed, as techniques: software licenses, recommended coding practices, and the public release of source code. In *addition* to these solutions, we believe that these challenges can be met by designing **tools**, **usability specifications**, and **features** for the software artifacts themselves. In short, we see a space for usability and design expertise for meeting these particular challenges for OSS.

Related Work: Usability and Open-source Software

So far, HCI scholars have been exploring how to inject usability expertise into the open-source development process [1,6]. They draw on the social and organizational aspects of open-source projects to explore how usability experts can participate in a development process where programming is a pre-requisite for participation and "code is currency". It is important to note that this related research shares a particular view of what such software is. They view "open-source" as a *method* of software development and as a type of *software license*. They do not otherwise appear to regard the software artifact as different than proprietary software. For example, their focus on the difference between OpenOffice and

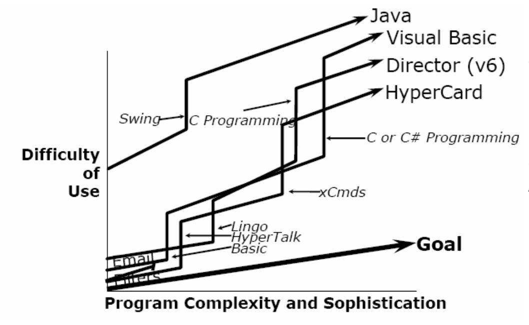

Figure 1. Difficulty vs. Program complexity & Sophistication

Microsoft Office would only be the process by which it was developed or the software license it is copyrighted under -- these elements aside, the two products could be interchanged. Our contribution to this area is a different approach. Our research question is to see how to make open-source products learnable, improvable, personalizable, and sharable – these are a different kind of usability than typical for proprietary software.

Related field: End-user Programming

We would like to encourage researchers in End-user Programming to consider the challenges and use-cases above as motivators for future work. End-user programmers are "people who write programs, but *not* as their primary job function. Instead, they must write programs in support of achieving their main goal, which is something else, such as accounting, designing a web page, doing office work, scientific research, entertainment, etc. [5]" This characterization matches our view of novice-programmers. There is a close fit between our challenges and the solutions that the field of end-user programming (EUP) is exploring. Major concepts from the EUP field help us to look for ways to make open-source software more learnable, shareable, improvable, and customizable.

For example, Brad Myers [5] uses a difficulty vs. program complexity and sophistication model to illustrate the way that programmers encounter barriers or walls that they must overcome in mastering a programming language. He identifies the goal of End-user Programming to be systems where learning occurs on a "gentle slope" instead of large walls of difficulty where complexity increases. We can use the same concept of "difficulty walls" to identify opportunities to make open-source software easier to study and to customize. One example of this can be found in Wordpress, an open-source web application for building public blogs.

Mastering Wordpress

The Wordpress web application is widely adopted; there exist over three million blogs that are built on this software. Although these different pages are built over the same Wordpress codebase, they are capable of a wide variety of functionality: custom accounts, calendars, visitor statistics, and more. Wordpress succeeds in providing a flexible framework through support for theme packages and plugin packages. This is necessary to fit the diverse needs of its users. Additionally, the amount of energy that is devoted to plugin-development and theme-development generate a community of contribution to the code-base that appears healthy for the main branch of Wordpress itself. Wordpress has a built-in interface for editing theme files and plugins from a web-browser. An administrator can use a web-browser to log into the administrator layer of his Wordpress blog and access a user-interface for changing themes, installing and uninstalling plugins, and even editing the PHP source-code for themes and plugins.

Following simple instructions, a user who is familiar with some HTML can use Wordpress to administer a blog. To leverage its powerful plug-in and theme support, he will need both a better understanding of Wordpress and more programming skills. Applying the graph in Fig. 1 to this situation, we might find "walls" of difficulty where our user needs to grasp the API, code architecture, and coding styles to master Wordpress.

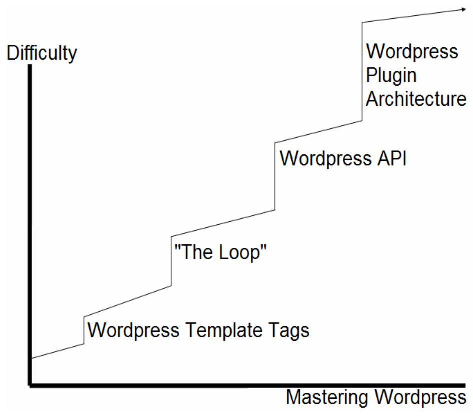

Figure 2. Potential learning slopes and walls for mastering Wordpress

A graph of these hurdles might look something like Figure 2. Suppose a user wanted to tweak a Wordpress template: change how a timestamp is formatted. This user would need to learn the template tags, a Wordpress construct for displaying dynamic information such as a post title, its author, or its timestamp. This is the first wall. If he wanted to do something more complex, like reordering the presentation of a post to show comments first, he would have to learn how Wordpress displays posts via a construct named "The Loop." This is another hurdle. While this mapping is a bit speculative (we haven't interviewed programmers), it demonstrates an analysis tool for recognizing *opportunities* to make an open-source project easier to study or to learn. End-user Programming offers solutions to these walls, many of which suggest new design directions for open-source software. The range is vast, from external debugging tools (e.g. the Why-line[3] [3]), better programming environments, to new programming languages. We appeal to EUP researchers to situate their research within the open-source contexts and use-cases that distinguish it as open-source software.

Summary

We described challenges for open-source software today: code reliability, customization, and novice-programmer userbases. We extracted a philosophy of software use from definitions of open-source software to show that "open-source" means more than publicly available source code. This philosophy is encapsulated in use cases: users studying the code, improving it, customizing it, and sharing it. These use cases are the solutions to our challenges. Solutions exist in usability, external tools and the architecture of the software itself. By addressing each use case in these respects, we can help tackle the larger challenges.

[3] Incidentally, Why-line was tested on open-source projects.

Current and Next Steps

We are exploring more related fields and areas that will promote the design and development of truly learnable, improvable, customizable, and shareable software – made easier. We have deployed an online survey to open-source communities to better substantiate our understanding of challenges to open-source projects.

References

[1] Bach, P. M., Kirschner, B., and Carroll, J. M. Usability and free/libre/open source software SIG: HCI expertise and design rationale. *Ext. Abstracts CHI 2007,* ACM Press (2007), 2097-2100.

[2] Fitzgerald, B. and Ågerfalk, P.J. The Mysteries of Open Source Software: Black and White and Red All Over? HICSS (2005).

[3] Ko, A. J. and Myers, B. A. Designing the whyline: a debugging interface for asking questions about program behavior. *Proc. CHI 2004*, ACM Press (2004), 151-158.

[4] Mockus, A., Fielding, R.T., and Herbsleb, J.D. A case study of open source software development: the Apache server. ICSE (2000), 263-272.

[5] Myers, B. A., Ko, A. J., and Burnett, M. M. Invited research overview: end-user programming. In *Ext. Abstracts CHI 2006,* ACM Press (2006), 75-80.

[6] Nichols, D.M. and Twidale, M. The Usability of Open Source Software. First Monday 8,1 (2003).

[7] The Free Software Definition – GNU Project – Free Software Foundation (FSF). http://www.gnu.org/philosophy/free-sw.html.

[8] The Open Source Definition (Annotated) | Open Source Initiative. http://opensource.org/docs/definition.php.