Open Source Basics: Definitions, Models, and Questions

Johndan Johnson-Eilola
Clarkson University
8 Clarkson Avenue
Potsdam, NY 12980
315.268.6488
johndan@clarkson.edu

ABSTRACT
In this paper, I will provide a basic overview of issues related to the use of open source models for development and distribution of computer documentation. The first section of the paper defines the key relations among different "open" categories (ranging from open standards to free software). The second section of the paper argues for two different methods for implementing open source models for computer documentation, one that offers increased user input into documentation projects and another that works to redefine how users and customers understand the importance and value of documentation.

Categories and Subject Descriptors
K.5.1 [**Legal Aspects of Computing**]:Hardware/Software Protection – *Copyrights*.

General Terms
Documentation, Economics, Legal Aspects, Management, Theory.

Keywords
Open Source software, free software, documentation, interface design, professional status.

1. INTRODUCTION
Open approaches to software development have generated an enormous amount of participation (and press) during the last several year [1, 6, 7, 9, 10]. Numerous incarnations of the "open" concept provide varying degrees of access to resources traditionally held as closed: open source, open standard, open framework, open participation, and more. Although "open" terms are sometimes used loosely, understanding the specific meanings of each is important because actual practices can vary wildly. For example, Open Standards initiatives may foster a sense of communal involvement in emerging standards. However, in some isolated cases such movements can be used to hijack public participation use. Conversely, companies that adopt common Open Source licenses may find that they unwittingly lose control over the direction their software or documentation is taken by subsequent adopters.

This paper includes two primary sections: an initial set of definitions and examples that points out key distinctions (and discussions of the rationale and implications behind each for general users and for technical communicators). In the second section, I'll attempt to frame some key overall ways that technical communicators might take advantage of (and landmines they might avoid) in considering open source projects (both developing documentation for open source software and adopting an open source model for some documentation projects).

Readers with a basic understanding of Open Source may wish to skip directly to Section 3.

SIGDOC'02, October 20-23, 2002, Toronto, Ontario, Canada.

2. VARIATIONS ON A THEME: DEFINITIONS AND RELATIONS

2.1 Introduction
One complexity to the Open Source movement has been the debate over what, specifically, the term itself means. In some instances, the gradations are so slight that they do not appear to have significant effects on most users and developers. The debate over "Open Source" versus "Free Software" between Eric Raymond and Richard Stallman [10], for example, turns on what seems like a subtle point (indeed, it almost sounds like a clichéd image of how committee meetings can generate mountains from molehills).

However, as participants in the debate attempt to make clear, "open source" attempts to front the increased reliability that results from large communities of users being given access to source code (a benefit that relates to the Open Source dictum, "with enough eyes, all bugs are shallow") while "free software" attempts to highlight how, in contemporary culture, software frequently structures how people act and communicate. In this respect, following the lead of theorists like Lawrence Lessig [5] open source advocates such as Tony Stanco point out that in online communities

> [S]oftware is the functional equivalent to law in real space, because it controls people, just like law does.... [it is] much more obedient and therefore dangerous in the wrong hands. [8]

In other words, "free," as Richard Stallman famously puts it, should be thought of as 'Free as in speech, not free as in beer" [10].

And if the Open Source versus Free Software debate includes some potentially important distinctions, other distinctions within this general area have larger implications for users and developers, in some cases with themes from open source being apparently highjacked reasons that seem to be at odds with much of the movement's overall goals.

Table 1. Key Open (and Related) Categories and Definitions

Term	Key Aspects	Major Names
Free Software	users can copy and redistribute	Richard Stallman
Open Source	users can access source code users can modify and redistribute code	Eric Raymond, Bruce Perens
GPL	Free Software Variation: Copyleft (hacking copyright law)	Richard Stallman
Open Standards	interface specs (protocols and frameworks) publicly available for use	IBM, Netscape, Microsoft
Shared Source	users allowed to see source code on as allowed by owners	Microsoft

Below, I'll briefly describe the main characteristics of each category before moving to the second (more important) section on the value of some of these elements to people in the field of computer documentation.

2.2 Free Software

Designed primarily by Richard Stallman, Free Software involves four key rights inherent for users, developers, and reprogrammers of software:

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

As Bruce Perens points out, Free Software has a relatively long historical status: computer programs were not initially sold on the open market, since no real market existed. Instead, programs were simply shared among sites (at least in some cases) [7].

2.3 Open Source

A small group of Free Software developers concerned that the term "free" was being misunderstood by both developers and users: the term had been defined to include a long list of specific provisions designed to allow modifications and redistributions by users (see previous definition). But because "free" is commonly understood as meaning, simply, "no cost", software that did not meet the Free Software definition might still be called "free".

Developed by Eric Raymond and Bruce Perens, Open Source software is sometimes called "a marketing program for free software" [1]. For nearly all purposes, programs that meet Free Software provisions also meet Open Source provisions. (The debate surrounding the two terms still continues.)

2.4 GNU Public License

The GNU Public License (GPL) developed as a category within the Free Software license. In Free Software, users retain the ability to modify source code they've obtained under the Free Software license, make modifications to the source code (to add capabilities, for example), compile, and the redistribute the resulting program under a new license (including proprietary licenses). This use is completely within the original Free Software license (cite:http://www.gnu.org/philosophy/categories.html#Non-CopyleftedFreeSoftware).

The GPL provides a licensing maneuver that hacks copyright: copyleft. Under the GPL license, all "improved" versions of software must also be released under the GPL license.

2.5 Open Standards

Open Standards are protections for the right to *implement* standards that are openly published. The Open Standards themselves are described by a governing body, then published. Individuals and corporations are then free to develop code that instantiates or works with the published standard. For example, basic HTML, as published by W3C, is an Open Standard; Internet Explorer, Netscape Communicator, Dreamweaver, FrontPage, Mozilla, Opera, and Galeon are all examples of programs (specifically, the imbedded browsers or preview applications within each) that implement support the HTML Open Standard. Notably, this list of programs work from different licensing models: Internet Explorer, for example, is a proprietary program that's freely available (but is not Open Sourced); Mozilla is likewise free, but Open Source. Dreamweaver's preview mode is not free and is proprietary. Whether or not any specific approach is preferable depends on personal politics, institutional affiliation, and numerous other local and concrete factors.

Open Standards are considered useful to the extent that programs and data from various organizations and individuals can interoperate more effectively. If, for example, a Web page obeys a specific version of HTML, different browsers can all display the page correctly because they have access to the standard describing how specific codes should be displayed on screen.

2.6 Shared Source

Shared source has generated debate in the Open Source community because, at one level, it appears to offer similar provisions as other variations of the Open/Free movement. But (as with many of the debates within this context), the terms in the license mean different things to different audiences. Although "shared" might be read by some as synonymous as "free", the

term actually means that the owners of the source code can offer different groups of users differential access to view the source code. That is, Shared Source is probably closer to a developers' partnership than an open community. In Microsoft's articulation of Shared Source, was designed to help software developers better understand the interactions of their own programmers with those of another developer. Open Source (and other) opponents to Shared Source point out that one goal of this license type is to provide the original developer with more accurate debugging data, since programmers from different companies can point to specific lines of code that are causing errors [4, 6]

3. Open Source Models for Computer Documentation

One might rightly ask why computer documentation professionals would give two hoots about the Open Source movement. After all, Open Source deals explicitly with source code rather than documentation. And many programs distributed under various open source variations are shockingly deficient in terms of documentation (often including nothing but a README file that covers program compilation and installation but not endues).

Open Source provides two very important opportunities for technical communication, one more or less functional (with important but relatively localized potential), the other philosophical (but with a potentially wide-ranging set of consequences for the profession).

3.1 Developing Open Source Documentation

The principle processes of Open Source map relatively well to the documentation development process. Although many linux programs come without professionally-designed, usable documentation, there do existing hundreds of programs for which free documentation has been developed by various users, shared freely, modified, and posted to Websites. In many cases, software originated by one writer is modified and reposted by subsequent writers according to Open Source licenses. But while there are a wealth of pieces of documentation available, most would benefit significantly from the attention of someone trained in computer documentation design. At the very least, the opportunity to modify and then redistribute such documentation could provide excellent learning opportunities for technical communication students as class projects. At a more ambitious level, this activity (and the others described in this section), if coordinated as a larger effort by an organization like SIGDOC, IEEE PCS, or STC (or a joint venture among these and similar organizations) would provide an excellent opportunity for publicizing the importance of technical communication to users.

Although to some extent computer program code operates differently that computer documentation, many of those differences are due to an artificial separation. For example, the increasing overlap between interface design and online documentation signals a tendency for documentation to overlap extensively with computer code. WinHelp files, HTML and XML, the design of software wizards and assistants all illustrate instances in which documentation is itself source code. And because computer interfaces themselves are often a primary method by which users learn how a program functions, Open Source software provides a key area into which technical communicators might gain the ability to learn and demonstrate the critical contributions that technical communication can make to software, beyond traditional documentation: the concepts of usability, communication theory and practice, audience, etc. can provide important tools in improving user experiences. But while technical communicators are often overlooked when it comes time to design interfaces (in favor of programmers or graphic designers, for example), Open Source software provides a natural testing ground for the skills of technical communicators.

For example, many popular linux programs possess little or no integrated online help. At a more immediate level, many programs (often the same programs) possess extremely rudimentary interfaces, or interfaces that appear streamlined but that pose significant usability issues for many users in their audience.

For example, one benefit of the linux image editing program The Gimp is its extensive use of discrete windows to display various controls and manipulation areas during image editing [3]. On launching the program on my own box, I'm greeted with the set of windows shown in Figure 1.

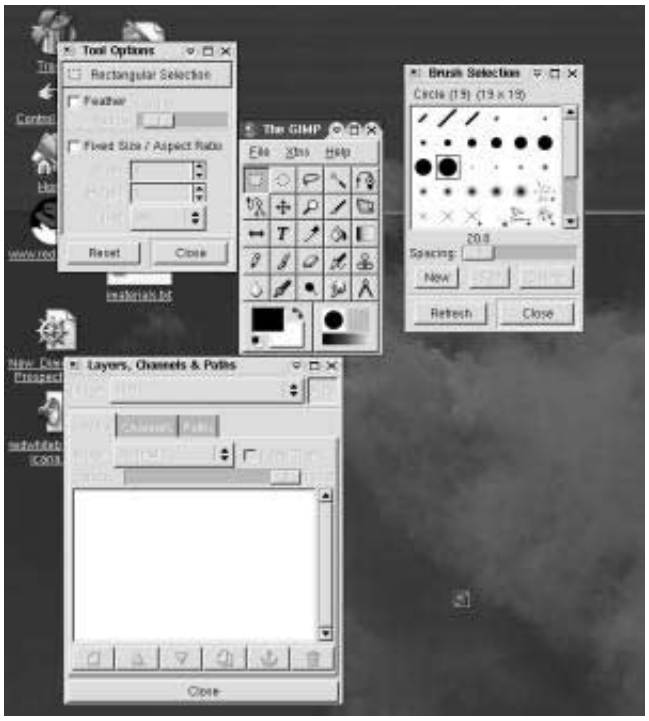


Figure 1: Portion of Screen After Launching GIMP

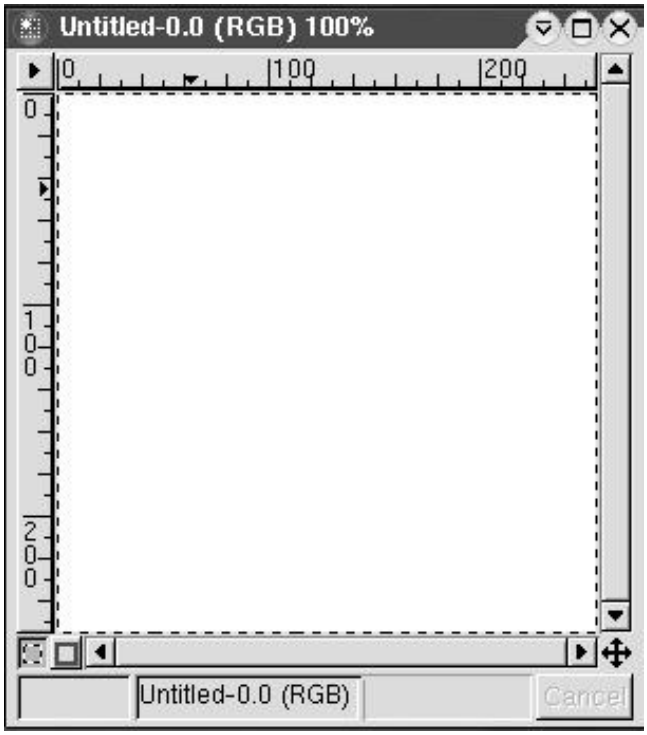
Each of these windows or palettes can be individually moved, resized, minimized, sent to other workspaces, etc. (I have not even open an image file at this point.) Like many linux applications, power users quickly develop efficient methods for managing screen information and can rely on memorized commands for moving screen information around to suit their particular contexts and patterns of work.

Intermediate users, however, may find that this interface lacks a coherent structure. For example, during editing sessions I frequently discover that all of my control palettes are either off-screen or hidden by other windows, leaving me with the set of interface cues shown in Figure 2.

From a usability perspective, intermediate (let alone novice) users are left with little clue about how what controls will let them manipulate the image window. There are no toolbox buttons, no menu headings or other hints about how to proceed.

Rhetorically speaking, this situation represents either a lack of understanding about complex audiences (that is, that there might be both expert and novice users who need support) or an intentional effort to obfuscate program interface elements in order to limit program use to power users.

Figure 2: Empty GIMP Document Window Lacking Usability Hints



In either instance, this situation provides an example of a program that someone with expertise in technical communication could provide relatively minor tweaks in order to improve the overall usability of a program for a wider range of audiences by applying general technical communication and usability techniques.

For example, in the GIMP window above, users need to right-click on the empty screen in order to call up a standard menu, a fact that many intermediate users (and above) but not many novices will know. Wouldn't it make more sense to provide a small menu bar even in small image windows? From a narrow audience perspective (one in which programmers are writing primarily to other programmers), no, since the menu would take up screen real estate. But given the current rhetorical goals of linux development (at least as stated by many in the linux community) to expand the community of users from a small core of programmers and hackers to a more mainstream desktop audience, providing additional cognitive scaffolding in the interface would be a good thing. Indeed, throughout GIMP (and with many other linux programs), there are hundreds of interface elements that could be modified according to standard principles of usability and communication in order to improve the usability of the program.

Certainly numerous programs, in open source or out, possess interfaces that could use the work of someone with a background in communication and usability. In fact, Photoshop itself has a notoriously steep learning curve. But because Adobe is not distributed under any variations of Open Source, Photoshop cannot be modified by end users, let alone redistributed. Under the Free Software, GPL version of the license that covers GIMP, these modifications could then be redistributed to other users.

Admittedly, the ability to edit source code is, in many cases, not a core competency for technical communicators. But given the number of technical communicators who work with one foot in computer science, a coordinated effort at program redesign by the discipline is not out of the question. And a single instance of successful redesign, worked on by a core team of technical communicators, could provide an important press opportunity for making visible the importance of technical communication skills (and an antidote to the recent spate of articles documenting difficult to use software).

3.2 Relocating Value from Programming to Communication

At a much more philosophical level, the open source movement provides a paradoxical shift in public understanding about the importance of programming: the ability to write computer code is not always of fundamental value in the late capitalist marketplace. In a sense, software is increasingly a commodity: in the Open Source model, people do not pay for mass produced and distributed software (although they do acknowledge the fact that it's important). Rather, they tend to value the ability to localize standard packages, with programmers at places using open source software being paid to customize applications to answer local needs.

Paradoxically, this movement does not necessarily act to depreciate the skills of technical communicators, even though it commonly does. As I mentioned earlier, documentation (and effective interfaces) for Open Source programs is frequently paid scant attention. But this situation exists only because we have not yet fully understood and capitalized on the Open Source shift. As many corporations have demonstrated, Open Source software development does not negate capitalist profit (despite Microsoft's accusations). Instead, Open Source tends to change the location of value: customers now value service organizations that solve problems. Red Hat, for example, one of the most successful linux distributors, generates much of its income (and mindshare) through training and certification programs. And O'Reilly Press, an extremely well know publisher of print books about linux (among other things) generates income through the sales of print books that are often available to users for free online. The movement to Open Source, in this sense, acts to place primary value on communication of one sort or another rather than on program function. The most profitable areas of the Open Source community are not in the development of Open Source software, but in the development of support materials, customized product integrations, training centers and materials, and certification processes.

This is precisely the niche that technical communication can fill: by leveraging understanding of usability, information design, learning theory, communication practices, and rhetorical theories, technical communicators possess key traits that would allow them to work within the Open Source movement, combining both Open and customized (for-profit) services at the same time.

4. REFERENCES

[1] "Advocacy." Open Source Initiative. Available at http://www.opensource.org/advocacy/. Accessed 6/20/02.

[2] "The Free Software Definition." The Free Software Foundation. Available at http://www.gnu.org/philosophy/free-sw.html. Accessed 6/24/02.

[3] GIMP. [Computer Software.]. Available at http://www.gimp.org/. Accessed 6/24/02.

[4] "Information on Microsoft's 'Shared-Source' Licensing." Share-Source. Available at http://www.shared-source.com/. Accessed 6/20/02.

[5] Lessig, Lawrence. *The Future of Idea: The Fate of the Commons in a Connected World.* Random House, New York, 2001.

[6] "Microsoft Expands Commitment to Open Standards and Interoperability." Microsoft Corp. Available at http://www.microsoft.com/PressPass/press/2001/Jun01/06-27CorelPR.asp. Accessed 6/24/02.

[7] Perens, Bruce. "The Open Source Definition." In C. Dibona, S Ockman, and M. Stone (Eds.) *Open Sources: Voices from the Open Source Revolution.* O'Relly Press, 1999.

[8] Stanco, Tony. "Information and Communication Technologies for Development: Lessons Learned and Directions for the Future." Paper presented at the World Bank's InfoDev 2001 Symposium, December 6, 2001, Washington D.C. Available at http://www.infodev.org/symposium2001/presentations/stanco.html.

[9] Wheeler, D. "Open Source Software/Free Software (OSS/FS) References." Available at http://www.dwheeler.com/oss_fs_refs.html. Accessed 6/20/02.

[10] "Why 'Free Software' is Better than 'Open Source'." GNU. Available at http://www.gnu.org/philosophy/free-software-for-freedom. Accessed 6/24/02.