October 2010

Discrimination in the Documentation of Open Source Software

Rebecca Claire Baron
Worcester Polytechnic Institute

Project Number: LDH-PW10

Discrimination in the Documentation of Open Source Software

A Major Qualifying Project Report
Submitted to the Faculty of the
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree of Bachelor of Science, Professional Writing

Submitted By:
Rebecca Baron

Advised By:
Professor Lorraine Higgins
Professor Jennifer deWinter

Submission Date:
October 27, 2010

Email:
Rebecca.C.Baron@gmail.com

Abstract

Open Source Initiative (OSI), the licensing organization of open source software, requires that the software "not discriminate against any person or group of people"; that is, open source software must be *accessible* to the average intelligent computer user. Some researchers, however, have suggested that access can be limited due to poorly written software documentation.

This professional writing study analyzed how accessibility might be influenced by the software's written documentation. The study defined access as whether an average user might *find, open, understand and use* documentation for the software.

Four users coded the documentation for 18 pieces of software on SourceForge.net to determine whether it existed and how easy it was to locate. Additionally, they determined whether it was static or dynamic and whether it employed technical writing practices noted for increasing user comprehension. Finally, coders shared their perceptions of the documentation that might affect usability.

This research indicated that documentation may be a key component in accessibility of open source software. A significant number of sites had no documentation whatsoever; moreover, while existing documentation employed many best practices for technical writing, they were not always implemented well. Two key problems may render the documentation discriminatory: 1) excessive undefined jargon in the documentation targets only specialized audiences, and 2) documentation itself is static--not controlled or modifiable by users, potentially constraining understanding and use of the software by larger audiences. More direct usability studies were recommended, and a list of recommendations was provided for writers of future documentation to ensure their writing does not limit accessibility. Finally, it was recommended that the OSI require some form of documentation and establish resources for creating dynamic and effective documents.

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

To my advisors, Dr. Jennifer DeWinter and Dr. Lorraine Higgins, for their endless support and for never giving up on me.

To Dr. Ross Micheals, my inspiration for this project, and to his new son, Zachary.

Preface

Dr. Ross Micheals, who works for The National Institute of Standards and Technology (NIST) discussed with me a project he had been working on—a new piece of software for biometrics. Biometrics is the science of the measurements of the body. It includes topics such as fingerprints and facial recognition. The software was to be open-source and to include new standards for fingerprint recognition. As a technical and professional writing major, I was interested in learning about the various types of documentation that go into a new piece of software, but when I asked about what was written for this piece of software, the answers he gave me were incredibly vague. He said, "There is some documentation, but it is not finalized," and "We have some notes," and other half-hearted responses indicating that the documentation for this new software was not a high priority for this project.

After doing some research, I found that other than a few release notes and paperwork internal to NIST such as proposals and whitepapers, very little documentation existed to accompany this new software. Further, what did exist did little to explain why the software exists and how it should be used. The first few sentences on the website read, "The NIST Image Group is announcing the availability of the NIST Biometric Image Software (NBIS), replacing the NIST Fingerprint Image Software (NFIS2) software package. The NBIS software is organized in two categries [sic]: Non-Export Control and Export Control" (NIST, 2006). These sentences, which seemed to introduce the software, meant absolutely nothing to me, and seemed hastily constructed and intended for a very advanced audience. Additionally, the obvious spelling error, while perhaps simply an oversight, suggested that NIST has designated neither sufficient time nor effort for documentation.

Based on my observations, it seemed that NIST assumed its target audience included companies and government agencies that already had a very thorough knowledge of biometrics. The documentation, especially the promotional and informative writing, was very limited and difficult to read. It was filled with jargon that might or might not be clear to someone who is already familiar with the field of biometrics. The opening page[1] was the first link listed when using Google to search for "NIST Biometrics Software." In spite of being centrally located and easy to find, it was difficult to understand. It was crudely formatted and filled with acronyms and technical jargon. The entire page contained almost forty sentences, only six of which had no acronyms in them. The text was small and required an existing understanding of the terminology being used and a high reading and comprehension level. NIST has made its website inaccessible to people who do not already meet these criteria.

NIST's target audience for their software, as made clear by its documentation, is a very technical group that is already very familiar with jargon relating to biometrics. The software is probably missing some potential users by being aimed at this specific target group. This might be a deliberate choice on the part of NIST or it might be an oversight caused by the programmers writing the documentation. Regardless of the cause, the lack of

[1] http://fingerprint.nist.gov/NFIS/index.html

clearly written documentation for a general audience seems to discriminate against many potential users. Since all open source software is required by the Open Source Initiative to be completely nondiscriminatory, this conundrum seemed perplexing to me and became the focus of the following project.

1: Introduction

The very definition of open source requires that software with that license be non-discriminatory. Yet, NIST's documentation does discriminate. Organizations that might choose to have biometrics in the future would probably find it difficult to use this software with its current documentation. As a result, these organizations probably would not use the software even if it could help them.

The problem of having documentation for open source software that discriminates against certain groups of potential users is not isolated to NIST. Many examples of open source software lack sufficient documentation, so NIST's biometrics software may not be unusual.

To understand the shortcomings of open source documentation, one first has to know about the goals of the open source movement. The stated goal of open source software is to offer software with flexibility, freedom, and availability, among other ideals. The intention is for users to be able to customize it via open source code—users having access to the code, which controls the program's actions. According to the Open Source Initiative (OSI), the governing body of all things involving open source licensing, "Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in" (n.d.). On the other hand, proprietary software, the "traditional" form of making software in which the user cannot access the source code, has a different set of ideals. By not allowing the users to access and modify the source code, the software's stability increases and the developers are able to make and sell modifications.

The documentation for proprietary software is often better developed than the documentation for open source software. This statement is incredibly broad and in no way applicable to all software, but it is a starting point for understanding how open source software might be discriminatory. Without adequate documentation, some potential users of open source software may be unable to find, use, or fully take advantage of the software. The discrimination in the documentation of open source software undermines the very ideals that open source software purports to hold.

In the traditional model for software documentation, as developed for proprietary software, the documents are created for a timely release with the software—a one-time occurrence. Documentation is only updated when the company updates the software and a new version is formally released. The company carefully orchestrates the entire process to ensure that all parts of the software package, including the documentation, are ready at the same time. When software patches[2] are released, the documentation usually stays the same.

With open source, however, the software is constantly undergoing changes by users. The documentation (when it exists) tends to be released by the software developers with a program's initial release. However, users can make modifications to the software and

[2] Patches are minor updates to the software that do not require a new version.

redistribute it without the initial developer's involvement. Problems arise if new documentation is made that does not reflect these updates. No single entity is responsible for maintaining or updating the documentation. Additionally, open source software has no formal quality control as opposed to proprietary software, which frequently does. As a result, there are often gaps in the documentation where changes have been made to the software more recently than changes to the documentation. An individual programmer might make a change to the software, but unless she also makes the necessary change to the documentation, the gaps become wider and more frequent. Therefore, because open source software cannot have a single person or group of people monitoring all changes, the users who make changes to the software must ensure that they keep the documentation up to date.

Figure 1 below simplifies the difference between proprietary and open source software. The diagram on the left represents open source software. By allowing the users to modify the software, the developers act more as moderators overseeing the users' changes and helping the technical writers to stay updated. If, however, the developer does not oversee the users' changes, or if there is no technical writer, then the changes made by the users are not documented. Compare this to the proprietary software diagram on the right, where the users have no impact on what the developers develop or the writers write. They simply use what is given to them. This leads to more stable software with up-to-date documentation because only one group of people is ever changing the software and its documentation.

Open Source Software Diagram Proprietary Software Diagram

Figure 1: Open Source versus Proprietary Software Diagram

The purpose of this project was to explore the extent to which the documentation that is used in open source software is discriminatory against potential users in its form, style, and presence or absence. I researched the differences between open source and proprietary software, invoked the research of scholars and software developers who have studied open source software and documentation, and performed my own direct research through an investigation of a sampling of open source software and their documentation.

Chapter Two contains detailed background information as well as a literature review of existing writings relating to open source documentation. I looked at the definition and history of open source software, a brief explanation of software documentation, and the reasons why open source documentation may be less than ideal.

Chapter Three describes my methodology for analyzing the documentation of eighteen different pieces of open source software.

Chapter Four reports the results of this investigation providing quantitative and qualitative information about open source software documentation and the extent to which it discriminates against potential users of limited technical ability.

Chapter Five summarizes my conclusions from the study and offers direction for other possible studies in the future related to the documentation of open source software.

2: Background

2.1: Open Source

Traditionally, software has been proprietary, and proprietary software does not have the source code—the code that is written so that the computer knows how the program works—available for the user of the software to view or edit. Most proprietary software does not let users change the code, and they must work within the limitations of the software as set by the developers. Open source software, on the other hand, allows users to make modifications to the source code in order to change the program to better fit their needs. This specific type of software has a bifurcated history contrasting social ideals with economic viability. Studying this history has allowed me to understand how open source developed and set the framework for my study of its documentation.

2.1.1: History

Open source software—software that has source code that a user can access and modify—has, in fact, existed for decades, although the term "open source" is relatively recent. Through the 1970s into the mid 1980s when software was "written" by punching holes in cards and feeding them into the computer, computer programs can be said to have been open source because a user could modify the program by adding or subtracting cards from the stack.

The modern open source software has two branches as shown Figure 2 below: the Free Software Foundation (FSF founded by Richard Stallman and the Open Source Initiative (OSI) founded by Eric Raymond.

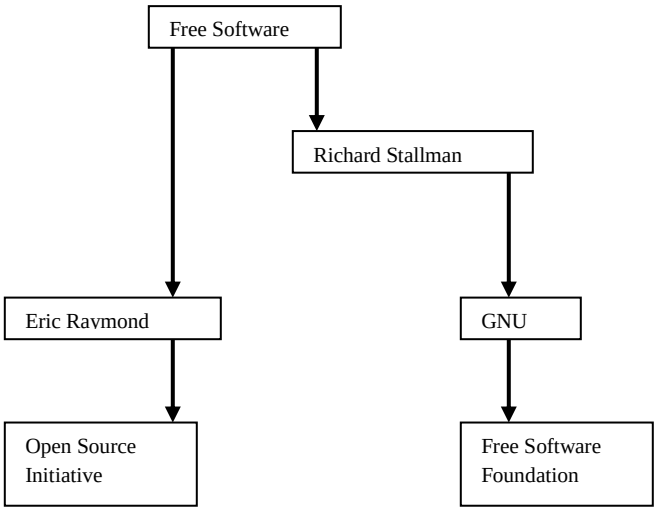

Figure 2: The Two Branches of Open Source Software

The two men agreed that the standard, proprietary method of creating and distributing software was suboptimal, but they each created a different solution. Stallman began his work on free software years before Raymond. As a programmer at MIT in the 1970s, Stallman is seen as the father of the open source movement as he was the first to draw attention to it. Stallman started the FSF in 1984 and influenced Raymond, a programmer previously of proprietary software, who later founded the OSI in 1998.

The split occurred due to differences in goals. Stallman wanted software to be free of restrictions so all people could use it. Raymond, on the other hand, saw open source software as an opportunity for financial gain. He wanted to remove restrictions so that anyone was free to earn money from this software. In short, Stallman wanted software to have had social freedom while Raymond wanted financial freedom.

The website for the OSI (n.d.) seems to indicate social goals, but careful reading shows that financial opportunity drives Raymond's desire for social reform. The opening page reads:

> Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.
>
> The Open Source Initiative (OSI) is a non-profit corporation formed to educate about and advocate for the benefits of open source and to build bridges among different constituencies in the open-source community.
>
> One of our most important activities is as a standards body, maintaining the Open Source Definition for the good of the community. The Open Source Initiative Approved License trademark and program creates a nexus of trust around which developers, users, corporations and governments can organize open-source cooperation. The OSI also describes itself as, "actively involved in Open Source community-building, education, and public advocacy to promote awareness and the importance of non-proprietary software.

The word "access" is notable in its absence in the statements of both organizations, but the words that are used by each shows that they value "accessibility" in spite of the fact that they have defined it differently. For example, the FSF uses phrases like "defend the rights of all users," and "the danger of not having software freedom." For the OSI, terms like "nexus of trust" and "build bridges" appeal to followers of Richard Stallman in their desire for social freedom, but Raymond focuses more on phrases like "lower cost, and an end to predatory vendor lock-in." In the case of the OSI, software "accessibility" seems to mean that any user may obtain the software and redistribute it, either with or without cost. The OSI is particularly concerned about distribution and circulation of software. For the FSF, however, "accessibility" seems to mean that users may use and modify the software without restriction. The OSI is most concerned with not limiting its community so that as many people as possible can access the software and redistribute it.

It is important at this point to discuss the issue of audience. Both the FSF and OSI want access for all people. Even though their definitions of access differ somewhat, their

goals create common ground. Is it reasonable, however, to expect access for all people, regardless of who they are? Should they be limiting their audience at all? For example, by saying “all people” both organizations include people who are not computer literate in their audience. People speaking every language, with every possible disability and of every age are included in the group of people who should have equal access to the software. This goal, as it is, seems unreasonable, so I assumed for this project that the target audience was all people who would want to use the software and that language and disability barriers can be handled within the software.

2.1.2: Richard Stallman and the Free Software Foundation

For a long time, everyone used Stallman's term "free software." The phrase "open source" did not appear until 1998 (Stallman, 2001). Stallman strongly believed that all software should be free of copyright and available to everyone. He wrote, “In the lab where I worked, the entire operating system was software developed by the people in our community, and we'd share any of it with anybody. Anybody was welcome to come and take a look, and take away a copy, and do whatever he wanted to do. There were no copyright notices on these programs” (2001). He was one of the first people to be vocal about software being available to everyone and modifiable by anyone. Stallman was used to working in an environment where anyone who wished to was able to find, open, understand, and use his software without restriction. The main obstacle for the FSF was that it was not publicized enough for people who were not already in the know to be aware that free software was an option. He did not apply any restrictions, but he did not seem to go out of his way to eliminate the restrictions in audience that existed already. Regardless, he was proud to offer open access to anyone and ended up devoting the rest of his life to promoting the accessibility of software.

Stallman began advocating for free software after an incident with a Xerox printer. The printer was a gift to his lab and, unlike the previous one Stallman had worked with, this printer used proprietary software. Stallman and his group had modified the code of the previous printer in order to include functions they found useful such as notifying the person who printed a job when it was complete. The new printer did not include that function and, since the software was proprietary, there was no way for Stallman to add it. After several conversations with Xerox, Stallman became discouraged and gave up on the printer. This incident prompted Stallman to fight for free software so that others would not also be discouraged by similar situations (2001). Shortly after this incident, Stallman founded an open source operating system that was humorously named GNU, standing for “GNU's Not Unix”. The inaccessibility of Xerox's software annoyed Stallman. In spite of how Stallman was able to find, open, understand, and use the software as it was, he wanted access to the source code as well, and that, Xerox would not grant. For Stallman, accessibility requires that anyone be able to use software *for any purpose* including ones not yet written into the software. If there were functions that a user wants, he or she should be able to add them. After his frustration with Xerox, Stallman dedicated his life to advocating for free software.

In 1984, Richard Stallman quit his job at MIT to devote his time to working on the GNU project, a Linux-like operating system designed to be entirely free software. Stallman emphasizes that "free software" refers to it being without restrictions and not without cost. He explains, "'Free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer'" (January 2010). In 1985, Stallman founded the Free Software Foundation (FSF) to oversee the GNU project as well as other, similar projects. As of 2009, Stallman was in charge of both the GNU project and the FSF as president.

Stallman[3] defines "free software" using four tenets, each representing a freedom that all users are granted. In his paper, "The GNU Operating System and the Free Software Movement", he explains:

> Free software possesses four essential freedoms:
> - You have the freedom to run the program, for any purpose.
> - You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)
> - You have the freedom to redistribute copies, either gratis or for a fee.
> - You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements. (January, 2010)

These four freedoms show how Stallman feels about each of the four aspects of accessibility. From these four freedoms, Stallman advocates that users be able to find the software through redistribution, open, and run the software, as well as modify and redistribute it. Stallman mentions nothing about users having the freedom to understand the software. Stallman wants software to be free and accessible to everyone, but by advocating freedom of finding, opening, and using without mention of understanding, he limits who may use the software to only those people who already understand.

2.1.3: Eric Raymond and the Open Source Initiative

Richard Stallman single-handedly led the free software movement until the late 1990s when the web browser Netscape chose to make its source code available to the public. Netscape brought programmer Eric Raymond in as a consultant. From his observations while working with Netscape, in 1998 Raymond wrote *The Cathedral and the Bazaar*, a groundbreaking look at the difference between open source and proprietary software and the processes for the development of each. In *The Cathedral and the Bazaar*, Raymond writes about the differences in development style and ideology between open source and proprietary

[3] As a side note to further understand Richard Stallman, a look at his personal website will show that he is quite the social idealist even outside of software. He has drawn cartoons criticizing the government for being oppressive, he has articles encouraging boycotts of products whose ethics he disagrees with, and he has other, similar activist musings. In fact, he describes himself as a "lifelong progressive activist" (n.d.). His work on the open source movement is simply one example of Stallman's desire for universal freedom.

software. He compares proprietary software to a cathedral where one person is telling you what to think and what to buy. On the other hand, open source software is likened to a bazaar with many voices shouting over each other trying to pull you in their direction, and you could choose to buy from them or sell your own wares (2001, p. 6). Raymond is very much in favor of the bazaar model for software as in it all people are free to sell their own goods as well as to buy from whomever they wish. There are no restrictions on who can make money on software. Within a year of publishing this paper, Raymond coined the term "open source" and founded the OSI to promote and advocate open source software. This was the beginning of a schism in open source/free software between Stallman and Raymond's interpretations and ideas.

2.1.4: Comparing the FSF and the OSI

Both the FSF and the OSI advocate and oversee the development of free and open source software. The two organizations, however, differ in many aspects. Most significantly, the two organizations have different goals for open source software. The FSF has primarily socially-oriented goals based on Stallman's definition of free software whereas the OSI strives for better software as a result of many people working on it. This is often referred to as the "many eyeballs" theory as it assumes that many eyeballs on a single project will result in a better product (Raymond, 2001). In spite of these differences, both organizations have advanced open source software and are important for its further development. Table 1 below is a chart showing the primary differences between the FSF and the OSI.

	Free Software Foundation	**Open Source Initiative**
Year Founded	1985	1999
Founder	Richard Stallman	Eric Raymond
Current Leader	Richard Stallman	Michael Tiemann
Goal	Freedom	Financial Viability
Terminology	Free Software	Open Source Software
Organization Described As:	Charity	Corporation
View on Documentation	Open Documentation License	Sell it to make money

Table 1: Comparing FSF and OSI

The simplest way to distinguish between the OSI and the FSF is to look at their ideals. The OSI wants open source software to be commercially viable while the FSF wants open source software to be free of restrictions. It is challenging to understand exactly what the FSF wants because it rarely uses a word other than "free" or "freedom" to describe its goals. The FSF seems to want as few restrictions as possible on software while the OSI wants to remove some restrictions while still have the software be financially viable. While both goals can be seen as championing access, the OSI is more concerned with people obtaining and redistributing the software while the FSF focuses more on the software's use.

The people who develop software do so for different reasons based on which organization they are affiliated with. The FSF appeals to people who want to work on free software for the political idealism, for hatred of Microsoft, and for the enjoyment of the process. The OSI, on the other hand, is more likely to want to create cheap, effective software to make an industry run more efficiently. They may also sell their software or the documentation for their software to make a profit.

Views on documentation are a very important distinction between the OSI and the FSF. The FSF states that the documentation is a part of the software so should be free just as the software is (FSF). They created the Open Documentation License to be able to apply the same freedoms to the documentation that are applied to the software, although documentation is in no way required for software. The OSI, on the other hand, suggests selling documentation as a way to make money (OSI, n.d.). Documentation is not included in the license and is not required to follow the same rules as the software. Not including documentation yet expecting software to be accessible is ignoring the "understand" portion of the access definition. If anyone may use the software but there is no documentation to help them to understand it, then access is limited. In particular, it means that people without the technical knowledge to be able to use the software without a manual are excluded and discriminated against.

2.1.5: The OSI Open Source Definition

Open source software is, by definition, free to obtain, modify, and redistribute by any user or potential user of the software. The Open Source Initiative (OSI) is the organization responsible for creating the definition and legal terms that determine if a piece of software is, in fact, open source. The OSI is a non-profit organization made up of programmers, legal experts, and other individuals who collectively monitor and maintain the open source license. According to the OSI's website, open source software is "software for which the original source code is made freely available and may be redistributed with or without modification" (OSI, n.d.). These three concepts, freely obtainable, modifiable, and redistributable, when implemented together, create the foundation of the open source movement. All three are critical for open source software, which identifies itself as being accessible and customizable for everyone.

Users can add, remove, and make changes to segments of the code so the program performs the functions specific to their needs. They can then choose to distribute their changes to other people. Many people choose to modify and redistribute software in the hopes that more minds working on a piece of software will result in a better product.

The vast majority of users do not modify the software, although most of them use versions of software that others have modified. It is difficult to determine whether users do not modify the software due to lack of knowledge of how to do so, or because the software already meets their needs. Regardless of the reasons why these users choose not to modify the source code, open source software still needs to have modifiable source code for all users.

The full definition of open source, as given by the OSI, has ten individual components, and all of them exist to require open source software to be equal for everyone. First and foremost, the software must be royalty-free to redistribute. The OSI's Open Source Definition states, "The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale" (OSI, n.d.). Since the OSI governs the open source license, that license cannot require royalties from anyone selling the software, although it does not restrict charging for the service of developing or installing said software. In his book, *The Success of Open Source*, Steve Weber (2004) defines "free" in terms of open source very effectively:

> The essence of open source software is that source code is free. That is, the source code for open source software is released along with the software to anyone and everyone who chooses to use it. 'Free' in this context means freedom (not necessarily zero price). Free source code is open, public, and not proprietary. (p. 62)

Developers can charge for software, but they must not require royalties for others seeking to modify the software's source code to redistribute it as their own.

Second, when distributing software that is open source, the developers of the software are required to make the source code for the program available. The Open Source Definition states:

> The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. Deliberately obfuscated source code is not allowed.

The source code being easily and freely available allows the user to modify the source code for personal use. In order for software to be open source, the user must be able to obtain and modify the source code without having to pay. David A. Wheeler (2007) writes in his paper "Why open source software / free software (OSS/FS, FOSS, or FLOSS)? Look at the numbers!", "[Open Source Software] users can tailor the product as necessary to meet their needs in ways not possible without source code. Users can tailor the product themselves, or hire whoever they think can solve the problem (including the original developer)" (p. 62). This aspect of open source is what differentiates it most from traditional proprietary software. Proprietary software does not allow users to modify the source code whereas open source software does.

The remaining rules in the definition of open source are intended to close loopholes that might be exploited. The licenses may not discriminate against any people, groups, fields, or endeavors. It also may not be limited to a specific set of technology (such as only work on a specific brand of computer), restrict other software (such as corrupting or blocking another

program), or be specific to a product (meaning that the license needs to be in effect regardless of how and in what package the software is distributed.) The license also restricts software from being distributed with a non-disclosure or other limiting clauses (OSI, n.d.). These rules, the ones that specify that the software may not discriminate, are most important to this project as they are the ones that allow all people to use the software. In particular, by using the word "discriminate," in spite of the fact that it is only used to refer to the license, an assumption is created that the software is supposed to be accessible to all people. It is this issue of accessibility that this project focused on, particularly in terms of whether the documentation allows less technical people to use the software.

2.1.6: Open Source in the Scope of This Project

SourceForge.net, the website that I used to find software to analyze, flags software as having "approved licenses" which are the licenses approved by the OSI. Second, the OSI is a more economically viable organization so is likely to outlive the FSF. Finally, the OSI uses the word "discrimination" in their definition, and while the definition does not apply to the documentation, if the OSI wants their products to be "nondiscriminatory" then they should be concerned with both the software and its documentation.

2.2: Documentation

Documentation is a necessary part of the software development process. In this section, I limited my discussion to software documentation. Software documentation as a category includes anything that is written down and describes the software. According to Ian Sommerville (2001), documentation can be broken down into two categories: process documentation and product documentation. Process documentation is anything written down about the way the software was made. This includes development schedules, proposals, reports, memos, comments within the code, and standard operating procedures. Product documentation is anything written about the product in its final form. This includes marketing materials, maintenance documents and documentation written for the end user (p. 2). I focused on product documentation targeted to the end user of the software.

According to Gerald J. Alred, Charles Birusaw, and Walter E. Oliu's book *Handbook of Technical Writing* (2006), end user documentation is the product information that whoever uses the product receives and includes several types of documentation. For software, end user documentation can include installation manuals, tutorials, user manuals, frequently asked questions (FAQs), help files, troubleshooting guides, and numerous other (p. 312). The overall purpose of these pieces of documentation is to assist the end user in performing certain sets of actions within the software. Installation guides help the user to install the program; troubleshooting guides help the user when the program is not working, and so on. Every piece of documentation makes it easier for the user to properly and effectively use the software.

Even before software was ubiquitous enough to require documentation, writers have been writing documents to teach another how to use equipment. As a result, many people have written about what makes good documentation.

2.2.1: Engineering Approach vs. Humanist Approach

Some say that having only basic information makes good documentation. According to Carel Jansen and Michaël Steehouder (1994) in their book, *Quality of Technical Documentation*, readers want, "an orderly, clean, clutter-free appearance, an obvious indication of what is being shown and what should be done with it, expected information located where it should be, and a clear indication of what relates to what" (8). Following Jansen and Steehouder's design leads to concise documentation that is easy to read. In her essay, "The triumph of users: Achieving cultural usability goals with user localization" Huatong Sun (2006) defines this method of documenting the "engineering approach" (460). The engineering approach is more product-based and has information limited so that the documentation is easy to read and very clean.

The problem with the engineering approach is that information is omitted. In order to write only the important information, someone needs to decide what is less important in order to select what is included. Another approach is to include all information related to the product and have the user search for the information that he would find useful. This is called the "humanist approach" (Sun, 461). This method of documentation makes it more challenging for the user to find what he is looking for, but ensures that everything is documented completely. The humanist approach does not give the documenters the power to decide what is most important. Instead, it gives the user access to all of the information and has him choose what he wants to read.

Most existing software documentation is written using the engineering approach because it is more efficient to write only what the developers think the users would want to know. Unfortunately, since the engineering approach causes information to be omitted, the users will sometimes be unable to find what they want to know, especially if they find themselves with an obscure question or problem.

The question of whether to use the humanist or engineering approach in writing technical documentation is an oft-debated topic with several ethical questions built in. Is it better to give the reader/user what they want or make them find it? If information is not included, who should decide what is omitted? Is efficiency more or less important than completeness? These are questions that Steven Katz (1992) addresses in his paper, "Ethic of Expediency." Many software developers answer this question for their own software not based on the ethics but based on the finances. It is less expensive to document only the "important" parts of the software. Were efficiency not an issue, technology allows software to be documented with all information being included, and allows the user to quickly and easily find the information she is looking for through use of a search bar or index (255).

Wikis, when fully filled in, are fairly humanistic. The user can find exactly what she wants and read that without worrying about the rest. Unfortunately, Wikis are very time

consuming to create and to populate with the complete information about the software. So, wikis are humanist when done well, but have the same problems mentioned earlier that they are time consuming and difficult to write without having the writers judge what is important.

2.2.2: Technical Writers

Some companies employ professional technical writers to write their manuals and other documentation. People can become technical writers many different ways including going to school, attending training courses, or starting from a writing or technical background and switching into the field. Regardless of how the technical writer reaches that position, she is trained in how to write the documentation for that company.

The documentation for open source software, like any other documentation, benefits from these suggestions, but often meets challenges not encountered by other types of documentation, particularly for other software. The biggest difference is that individuals or small groups develop the software as opposed to companies that develop commercial software. As a result, professional technical writers do not write the documentation. Instead, the software developers do most of the documenting of the software. One problem that arises from this is that developers assume that all readers have the same level of knowledge as a software developer and write accordingly. This is one of the roots of the issue of discrimination in open source documentation. The developers use the engineering approach for writing the documentation, but filter out information that non-developers would find necessary.

In general, good documentation, commercial or open source, has the following properties:

- Is written for an intelligent but uninformed audience
- Is well organized
- Has a neat appearance
- Is task-oriented
- Uses strong verbs
- Uses numbered and bulleted lists
- Use images and diagrams

<u>Written for an Intelligent but Uninformed Audience</u>

The problem mentioned before of software developers writing documentation at too technical a level for users to understand is the result of them writing for an audience that they assume to be informed. It is important, however, to write documentation with the assumption that the reader is intelligent but has no prior knowledge of the topic that the documentation is written on. Matt Young (2002) explains in his book, *The Technical Writer's Handbook*, the importance of recognizing the audience as having less experience and specific knowledge than the writer. He writes, "Write for the uninformed reader. Speak to an intelligent and sophisticated but relatively uninformed audience" (p. 63). If technical writing for open source

software as a whole were to follow Young's advice with regard to audience, the genre would be closer to being universally comprehensible.

Well Organized

On a stylistic level, there are some traits that are commonly used to define good documentation versus bad documentation. In his article "How to Improve User Guides," Ivan Walsh (n.d.) lists the important aspects of documentation. He says, "Well-written documentation should be easy to: Read, Understand, [and] Access." Without any one of these, the usefulness of documentation diminishes. If the users cannot access, read, or understand it, they cannot be expected to learn from it. Many different people have suggestions for how best to organize or write user manuals, but most directly relate to Walsh's three ideals for documentation.

Is Task-Oriented

Since the primary function of user manuals is to teach users to perform tasks with that which is being documented, the writing style should reflect that. As David A. McMurrey (2001) explains in his book, *Power Tools for Technical Communication*, manuals should be organized by the tasks they are instructing. McMurrey advocates the use of numbered lists. He writes, "Instructions in user guides should generally be task-oriented—that is, written for specific tasks that users must perform. Instructions should generally use vertical numbered lists for actions that must be performed in a required sequence. Similar or closely related instructions in user guides should be grouped into chapters" (128).

Uses Strong Verbs, Lists/Bullet Points, and Images and Diagrams

Writing handbooks such as the Gerald J. Alred, Charles Birusaw, and Walter E. Oliu's book *Handbook of Technical Writing* (2006) have very specific suggestions for writing good documentation. This book suggests using lists of numbered steps or bullet points instead of paragraphs of prose to make it easy for a reader to follow instructions. Additionally, the copious use of images as a replacement for lengthy explanations makes documentation easier for a user to follow (pp. 525-526). Numbered lists, bullets, and images make documentation easier to read and understand by simplifying the information from long blocks of text. The assumption is, however, that the images and lists contain useful, relevant information and, that they add to the documentation instead of distracting from it.

Images are most useful when they are clear, straightforward, and well labeled. An image (such as a chart, diagram, screenshot, flowchart, etc) is not useful if the reason for its presence is unclear. Also, the image should be close to the text that it is intended to clarify. As important as images are, language is also very important. Good documentation uses straight-forward language with imperative, simple verbs in order to make the language as clean as possible. When using lists of steps or instructions, each instruction should start with an imperative verb. The verb should be of as simple a vocabulary as possible.

2.2.3: Technical Writing in Engineering

Even when using the engineering approach, certain components should be included. In order to try to ensure that software documentation maintains some consistency, the Institute of Electrical and Electronics Engineers (IEEE) has defined a set of standards that, when followed, ensures that the documentation is readable. The standards are completely optional, but following them allows a software company to have guidelines to know if their documentation is sufficient. Below is a sample table showing requirements provided by the IEEE standard.

Each component listed in the Table 2 below is important. The identification data lets the user know what is being documented. The table of contents, list of illustrations, navigational features, index, and search capability make it easier for readers to find the information he is looking for. The introduction, information for use of the documentation, and concept of operations provide meta-knowledge that helps the user to best apply the documentation. The procedures, information on software commands, and error messages and problem resolution sections are where most of the information that the user needs would be. Finally, the glossary and related information sources teach the user more about the subject of the software if he is interested and could not find a certain piece of information within the documentation itself.

Component	**Required?**
Identification data (package label/title page)	Yes
Table of contents	Yes, in documents of more than eight pages after the identification data
List of illustrations	Optional
Introduction	Yes
Information for use of the documentation	Yes
Concept of operations	Yes
Procedures	Yes (instructional mode)
Information on software commands	Yes (reference mode)
Error messages and problem resolution	Yes
Glossary	Yes, if documentation contains unfamiliar terms
Related information sources	Optional
Navigational features	Yes
Index	Yes, in documents of more than 40 pages
Search capability	Yes, in electronic documents

Table 2: Components of Software User Documentation--IEEE Standard 1063

Software developers are not trained in technical writing so are unlikely to follow standards such as the IEEE standard. This can cause pieces to be missing or of poor quality.

Open source software is required to be nondiscriminatory. Since not everyone has the same amount of technical knowledge, good documentation can allow someone of limited computer experience to use the software that would otherwise be too complicated. Poor

documentation, however, does not make complex software any easier to understand so does not reduce discrimination.

Much of the IEEE standard is a good example of information mapping, a style of organizing information for ease of understanding. Robert E. Horn (n.d.), an expert in information mapping, provides an overview of the process:

> Information mapping is a method of bringing together current learning research and instructional technology into a comprehensive materials development and presentation technology to improve technical communication.
>
> A system of principles and procedures for
> - identifying
> - categorizing
> - interrelating and sequencing, and
> - presenting graphically information required for learning and reference.

One main concept in information mapping is "chunking." Chunking is a technical term meaning grouping similar ideas and concepts together so readers can easily find what they want to read. It sounds somewhat intuitive, yet effective organization of similar topics can make a big difference between effective, easy to read documentation and difficult, discriminatory documentation.

2.3: Discrimination

A recent trend in products including software and the documentation thereof has been universal design, designing a product for use by anyone. According to Gregg C. Vanderheiden (1996) of the Trace R&D Center at the University of Wisconsin-Madison, Universal Design is defined as, "the process of creating products (devices, environments, systems, and processes) which are usable by people with the widest possible range of abilities, operating within the widest possible range of situations (environments, conditions, and circumstances)." The idea is that, with universal design, anyone can use a given product regardless of circumstances or ability.

Universal design is typically used to design products that people with vision, hearing, cognitive, or physical disabilities can use as easily as those without. For example, websites that have an option of having text being read, manuals with pictures as well as writing, oversized buttons on remote controls, and voice commands on a cellular phone are all ways that people design products to be as universally accessible as possible.

The reason to include universally accessible features in a product is to demonstrate commitment to the user. Even if features are included for only a small percentage of users, they can also recommend the product as easily as those who do not need the universal design features. Additionally, universally designed features may make it possible for one group of people to use the product but may also make it easier for members of other groups as well.

For example, voice activation for cellular phones is necessary for people who are visually impaired but also used for people when driving.

Even if specific features are not implemented for universal design, considering the concept when designing a product can make a huge difference. For open source software where accessibility is a requirement, having a universally designed manual or other documentation allows a piece of software that would otherwise be inaccessible to people who are less technical to be used regardless of technical knowledge. Supplemental manuals that are universally designed are currently available for a variety of products (the *Idiot's Guide To…* Series and the *…For Dummies* Series are two examples), but information that anyone can understand should be available with the product itself.

2.4: Is Open Source Documentation Discriminatory?

Both commercial and open source software need documentation in order to be efficiently developed, distributed to, and usable by interested users. Unfortunately, a large percentage of open source software does not have enough documentation of good enough quality to allow the software to maximize its effectiveness.

Jack Herrington (2003), staff writer for DevX.com, conducted an informal study of the documentation provided for open source software. He looked at the top twenty projects on sourceforge.com—a popular database of open source software—and compared the documentation available. He saw that all of the software he looked at had a statement describing the software but only two percent had a statement of what problem the software solved. Half of the software he looked at had an FAQ, half had a tutorial and fifteen percent provided system specifications. Herrington asks in his analysis, "Is [the documentation] only intended for engineers already using the tool?" These percentages each represent the number out of the total that included a given type of documentation, although the report did not discuss the extent to which the software that includes or lacks given types of documentation overlaps.

Why does open source software lack sufficient documentation? Many factors contribute to the discrepancy between the documentations of open source and commercial software. The first is that open source software changes much faster than commercial software and the documentation frequently cannot keep up. This is particularly true for the larger open source programs. Robert Nagle (n.d) explains in his online article, *Does Open Source Documentation Suck*, "The problem is that most open source software is updated fairly quickly, and that web documentation may not be relevant to the version installed on your system". Open source is designed to be able to be changed and redistributed, but without a central group of people overseeing the changes, the documentation is not always updated for every version. In particular, changes made and redistributed by users frequently are left under-documented or undocumented.

These are only the most common problems with open source documentation, yet it is not hard to see that the way things currently work is less than ideal. In fact, current open source documentation causes many pieces of software to border on discriminatory. In spite of

the fact that rule number five of the definition of open source states, "No Discrimination against Persons or Groups," large numbers of people find themselves unable to use the software to its full potential (OSI, n.d.).

The fact that users are making modifications to the software and redistributing it causes other problems in documentation in addition to a slow update rate. With commercial software, people who are trained in technical writing write the documentation. When open source software is written or updated by programmers with no technical writing training, the idea of documentation might slip their mind or the documentation that is written is less likely to be complete or understandable by a general audience. Elena Blanco, writer for Oss-Watch.ac.uk (2008) explains, "Writing documentation requires a specific set of skills that are not commonly found within open source development communities." Not all open source programs have this problem, but it is more prevalent in open source than in commercial software.

Another common issue is that the people who write technical documentation for open source software underestimate the breadth of their audience. They might assume only a specific, small group of people will be interested in their product, yet they post it on the Internet. Since open source software is modifiable, someone outside of the perceived audience might stumble upon the software and want to use it for a slightly different purpose than its intension, but the documentation is so limited that he is unable to do so.

Furthermore, the very nature of open source software presents some problems with documentation. Writing may be available for how to use the software but there frequently is nothing written about how to modify the software. This effectively divides the user base for a piece of software into three groups: those who would never want to make modifications, those who can and do make changes to the source code, and those who would make modifications if they knew how.

The rationale for rule five posted on OSI's annotated list of rules reads, "In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process" (OSI, n.d.). While discrimination on the basis of technological capability is rarely as big of an issue as other forms of discrimination, the lack of proper documentation for open source software causes people who are less familiar with computers to be unable to use the programs or make modifications to the code. In short, they are locked out of the process.

To summarize, challenges to open source documentation include:

- Written by amateurs
- Low frequency of updates
- Lack of communication between developers
- Lack of centralized development team
- No clear line between developer and user
- Underestimation of breadth of audience
- Overestimation of technical level of audience

3: Methodology

3.1: The Study

My study investigated if and how open source software discriminates through its documentation. I wanted to answer the following questions: To what extent is the documentation of open source software discriminatory? In what ways might it be discriminatory? Could it be discriminatory in ways identified by prior research?

In order to address these questions, I selected eighteen different open source programs from the collection of software referenced on SourceForge.net, a leading database of open source software. I asked 4 volunteers[4] to code each piece of software noting whether documentation existed, indicating the ease of accessing and opening it, and marking whether the documentation adhered to a list of best practices for writing documentation. I also asked them to rate the extent to which they perceived the documentation as being logical, well organized, easy to read, and frustrating. A copy of the coding sheet as it was distributed to my coders can be found in Appendix B.

Before coding began, I talked the coders through the coding sheet and answered their questions. I was also nearby to answer questions as they arose while they were coding. I was careful to keep the answers as neutral as possible and not bias their responses when answering questions. . They primarily asked for clarification as to what the coding categories were.

Each website was evaluated by one coder and me. Each coder looked at 2-5 websites, depending on how long each website took. Websites that had no documentation took very little time while those with extensive documentation took longer. I then compared the coder's completed sheets with my own for each website to evaluate for consistency of results. Only 3 sites out of 18 (17%) had discrepancies. I resolved these discrepancies by interviewing the coder to verify that she had coded as she intended to and to learn more about the rationale for her response. I chose to defer to the coder for discrepancies because my background knowledge of the topic of software documentation could have caused bias in the results. In all three cases of discrepancy, the interview provided the clarification necessary to determine the result to be recorded; two provided insight to their responses, and one pointed out an error in her coding. If the discrepancies could not be resolved through interview, I was prepared to ask a third coder to look at the website and resolve the dispute based on the most frequent response, but that was not necessary

[4] My coders all have basic computer literacy and are "intelligent but uninformed." Susan and Fred Baron, my parents, both are graduates of the University of Pennsylvania. My sister Jacqueline and her roommate Stephanie Greenlaw are second-year students at Smith College.

3.2: Selection of Programs

Given the huge number of open source programs available on SourceForge.net, I developed very specific criteria for selecting the software I studied. First, all the software in the study have released files. Some software projects use SourceForge.net to store projects in progress that have not yet created a released program. All of the projects I selected, however, have at least released a complete draft of the program so that any documentation that exists can be used to understand an actual program and not just a concept.

The second requirement was that all of the projects use open source licenses that have been approved by the Open Source Initiative (OSI) (the governing organization for software using the open source licenses). The reason for this requirement was that the software had been approved by OSI and, therefore, must have met the requirements of the open source definition, including the nondiscrimination clause. There is no doubt that these programs can be considered open source software as defined by the OSI. They are all held to the same standard of nondiscrimination.

Programs were also selected with diversity in mind. The purpose of selecting very diverse programs was to be able to look at a variety of open source software and to generalize across differences. Using the genres of software as defined by SourceForge.net[5], I chose my eighteen programs by selecting two pieces of software from each of nine different genres. Genres define the category of purpose of the software. For example, genres include games, text editors, office/business, etc. SourceForge.net uses 20 genres, but many genres did not have enough software meeting my other criteria to include in the study. This limitation made my sample size somewhat small and not necessarily representative of all open source software, but provided some diversity.

My goal was to choose software with which potential users would be unfamiliar. In doing so, I tried to choose software for which the documentation would be particularly critical. All of the programs selected are self contained or web-based as opposed to additions to applications users might already be familiar with. I chose programs that meet this criterion to minimize the expected prior knowledge from potential users. If the programs were add-ons for other applications, then a certain amount of understanding of the original program would be expected. Additionally, some information might be omitted from the documentation because it was in the documents for the original software.

SourceForge.net's ranking for software was an important factor in my selection of programs. SourceForge.net ranks software by assigning each program a consecutive number based on the number of times the program's site has been visited, the number of times the software has been downloaded, and the frequency with which the software is modified. The software that is ranked "1" is most actively used and modified whereas programs with higher

[5]The software fit into the following genres: Communications, Games/Entertainment, Internet, Multimedia, Office/Business, Scientific/Engineering, Software Development, System, Text Editors. There were two pieces of software from each genre in this study.

numbered rankings are active. There are almost 400,000 projects listed and ranked on SourceForge.net.

I also specifically selected programs that are not widely known, so my coders would not have preexisting knowledge that might have influenced their views about the documentation for the software. I did this by rejecting any program ranked in the top 1,000 on SourceForge.net. This meant the survey answers given were based solely on the participants' observations from the documentation.

Additionally. I did not want to study software that had been abandoned or that otherwise was unfit to study. To prevent this, I chose software ranked better than 25,000. By limiting myself to the better-ranked software, I had a greater likelihood of studying currently active programs instead of ones that had failed in the past or had been abandoned.

Given the small sampling size, my conclusions are tentative and exploratory.

In summary, I chose the eighteen programs for my study using the following criteria:

- All programs are found on SourceForge.net
- All programs have OSI approved open source licenses.
- All programs are ranked higher than 1,000 and lower than 25,000
- There are two programs from each of nine genres

For the complete list of software used in this study, see Appendix A.

3.3: Features that were coded

According to the Open Source Initiative (OSI), open source software is required to "not discriminate against any person or group of people." Unfortunately, in many cases, open source software may have documentation that seems to discriminate against users of limited technical knowledge. It may discriminate by being written at too technical a level, by not being up to date with the software and, in some cases, by not existing at all. The study answered whether and how the documentation for eighteen different open source programs might discriminate against a general audience. It tested the documentation's accessibility based on whether my coders could **find** the documentation, **open** it, **understand** it, and **perceive it as usable**. The study assumed "an intelligent and sophisticated but relatively uninformed audience" as described by Matt Young (2002) on his website. This means that the target audience of the documentation was assumed to have sufficient understanding of computers to be able to use the Internet to find the software, download and install the software, navigate menus, and follow written directions as all of my coders did.

In what follows I have listed the questions used on the coding sheets and explained how they measure access in the documentation for the software. The actual coding sheets are in Appendix B.

Finding and Opening Documentation

Is There User Documentation?
Yes/No

User documentation, for this question was any sort of documentation for the user of the software (as opposed to the developers, marketers, etc.). If the answer to this question was "No" then coding terminated, as the rest of the questions asked about details of the documentation.

In most cases, a piece of software that does not have documentation discriminates against people without the knowledge or confidence to use the software without written guidance. Software developers may claim that the software is intuitive and does not require documentation. Even in these cases, though, documentation is beneficial, especially since content that a developer considers to be intuitive may be challenging for a user.

How Easy Is the Documentation to Find?
This section refers to how easy it is to find the documentation on the software's website. Documentation that is easy to find is more accessible than documentation that is deeply buried on the website. Users become frustrated when they must spend too much time digging around a website for the information they seek.

How many clicks does it take to navigate from the home page to the documentation?
Easy (0-3 Clicks) Moderate (4-6 Clicks) Difficult (7+ Clicks)

The first metric for quantifying how easy the documentation is to find is number of clicks from the opening webpage. Number of clicks quantifies how deeply buried the documentation is in the website. More accessible documentation can be reached in fewer clicks. If the documentation is on the opening page, it can be accessed in zero clicks. If it is linked to from the opening page, it can be accessed in one click. If it is linked to from a page that is linked to the opening page, it requires two clicks, and so on. This is frequently referred to as "click depth".

Thomas Powell (2000), a web designer who has written books on creating effective web pages, cautions in his book, *Web Design: The Complete Reference* not to have a click depth of greater than six. "Aim for a site-click depth of three. The three-click suggestion makes sense when considering the limited number of locations for different navigation bars on pages, traditional GUI conventions, etc." (p. 111). The reason for this is to keep users from having to navigate many pages

of menus. Web designers Sarah Horton and Patrick J. Lynch (2009), add in *Web Style Guide*, "Users should never be forced into page after page of menus if direct access is possible" (p. 79). If the webpage had a click depth of three or less, it was considered "easy" to navigate while more than six was "difficult." If the click depth was four or five, it was "moderate."

Coders were instructed to disregard use of the "back" button in the web browser and to start all counts from the opening page. If there were multiple routes to the documentation from the opening page, the coder recorded the lowest number of clicks.

Can documentation be found using a search bar?

Yes/No

Can documentation be found using a site map?

Yes/No

Search bars and site maps are two specific ways that websites allow users to access content that otherwise may be hard to find. Including these two features and allowing users to find documentation using them makes the documentation more accessible. Search bars are particularly important when a website becomes expansive. Horton and Lynch (2009) write, "If your site has more than a few dozen pages, your users will expect web search options to find content in the site" (p. 79). The inclusion of a search bar or site map are independent of click depth but are also important features that improve accessibility.

Location of Documentation Link on the Page:

Place an X in the box where the link to documentation is located.

This question addressed the issue of how easy it was to find the link to the documentation once on the correct page of the website. A link located in a menu of other links is easier to find than a link in the middle of a block of text. Also, by looking at many possible locations for the links to the documentation, I saw if there was a common location on the website where people can expect to find this information. "Users have developed clear expectations about where common content and interface elements are likely to appear" (Horton and Lynch, 2009, 92). For example, users expect internal navigation on the top and left of the page, while they expect external links to be on the right and bottom of the page. A "help" button, which often links to documentation, is usually expected to be near the top right corner of the webpage.

Formats and Genres of Documentation

What Format(s) Is/Are Used for the Documentation?
Select all that apply:

- Web page
- Web 2.0 (e.g. Wiki, Blog, Forums, etc.)
- PDF
- Word document/.doc
- Other_____

What Genre(s) Is/Are Used for the Documentation?
Select all that apply:

- Unlinked Document (Document without links to other parts of the document or outside resources)
- Linked Document/Hypertext (Document with links to other parts of the document or outside resources)
- FAQ (Document formatted as questions with answers)
- Forum (Online community where people post questions and responses for general use)
- Wizard (A dynamic, computerized guide to completing tasks which prompts the user to select relevant information)
- Searchable Database (Computerized system requiring the user to use a search bar to obtain information)
- Video tutorial (A video showing how to complete tasks)
- Wiki
- Blog
- Other _____

The format and genre for the documentation is incredibly relevant as it determines how the document is read. A PDF is static and cannot be modified while a web 2.0 format is dynamic, allowing the reader to also write to it and make changes. Additionally, documents such as Wikis are more likely to be written using the humanist approach than static documents which usually use the engineering approach (Sun, 2006, 460). With dynamic documentation, people can ask for the information they need (in documents such as forums) and find specific pieces of information to suit their needs. For static documentation, however, the user relies on the information provided by the initial writer but cannot influence what information is provided. In a dynamic field such as open source software, static documentation limits users' access by making it hard for people to change the documentation as the software is modified. Static documentation decreases the likelihood that the documentation will be up-to-date with the current version of the software.

Understanding Documentation

Technical writing experts have identified a number of best practices that help ensure readers will understand the material being covered in the documentation. In this section of the coding, I looked at the websites with documentation to see which ones applied the recommended best practices to be more understandable and, therefore, more accessible.

Who Is the Target Audience of This Documentation

Computer industry/community (i.e. software developers, IT, etc.)
Other industry/community (i.e. hotel owners, dog owners, etc.)
Other specific audience: Write in.
Mass Use (no specific audience)

Technical writers must have a good understanding of their audience when they write documentation. On his website, Robert Bly (n.d.) implores writers, "Know your reader—Are you writing for engineers? managers? technicians? lay people? Make the technical depth of your writing compatible with the background of your reader." Bly, however, was writing about traditional documentation instead of open source. The dynamic nature of open source software changes the traditional view of audience. The software's target audience may include people who have no interest in using the software as it is, but want to modify the software into something different. As a result, documentation that is not targeted to a mass audience discriminates against potential users of the software.

Is there Jargon?

Yes/No

Select one page. On that page, how many jargon words/terms are there?

Jargon words are: (Select one)

Industry/Community related? Software related? Both?

Is there a glossary?

Yes/No

Is jargon defined using parenthetical definitions?

Always Usually Sometimes Rarely Never

Is jargon defined in the margins or in footnotes?

Always Usually Sometimes Rarely Never

Is jargon defined using hyperlinks?

Always Usually Sometimes Rarely Never

To count instances of jargon, each coder selected a different page in the documentation. The jargon counts were then averaged for each website. As the base unit of documentation is the word, if the reader does not understand the words used, she will not understand the documentation. Unfortunately, complicated language and jargon are very common in technical documentation. Robert Bly explains, "Technical writers sometimes prefer to use big, important-sounding words instead of short, simple words. Technical terms are helpful shorthand when you're communicating within the profession, but they may confuse readers who do not have your special background." If the software is targeted at a specific industry, then industry jargon will likely be included as it will be understood by the readers. Technical or software jargon can also detract from the ease of understanding Gerald J. Alred, Charles Birusaw, and Walter E. Oliu's (2006) book *Handbook of Technical Writing* says, "If all your readers are members of a particular group, jargon may provide an efficient means of communications. However, if you have any doubt that your entire audience is part of such a group, avoid using jargon" (p. 288). The jargon used helped the coders in the previous question determine if the documentation was only intended for the target audience of the software. If jargon was not used frequently or was always defined, then the documentation's target audience extended beyond that of the software.

Sometimes technical jargon is inevitable. In these cases, though, the jargon should be defined, either in the text or in a glossary. Desmond D'Souza and Alan Wills (1998) write in *Objects, Components, and Frameworks with UML: The Catalysis Approach* "In addition to a narrative, it is useful to have an index of the vocabulary. The glossary's purpose is to link the formal terms back to the real world" (p. 190).

Is the documentation written in the 2nd person? (Imperative or "you")

Always Usually Sometimes Rarely Never

Does the documentation use culture- and gender-neutral language?

Always Usually Sometimes Rarely Never

Does the documentation use correct grammar, punctuation, and spelling?

Always Usually Sometimes Rarely Never

Does the documentation use the active voice?

Always Usually Sometimes Rarely Never

Does the documentation use the present tense?

Always Usually Sometimes Rarely Never

Does the documentation begin instructions in the imperative mode by starting sentences with an action verb?

Always Usually Sometimes Rarely Never

A great deal of research has been conducted on how to write clear, understandable instructional materials. The literature identifies many "best practices" involving the use of jargon[6], proper English, images, and organization. I based the above list on "accepted style standards" written by JoAnn Hackos and Dawn Stephens (1997) in *Standards for Online Communication* (pp. 268-273). These questions all addressed the language of the document based on established guidelines for writing documentation. These guidelines are not specific to software documentation, but documentation that follows them is generally easier to read and understand than that which does not. Clear documentation is easier for all readers to understand and, therefore, is more accessible.

Are images used?

Yes/No

How many images are in the document?

Are drawings used?

Are they used to show objects and spatial relationships?

Did you identify an area where there should be a drawing?

If yes, where?

Are maps used?

Are they used to display geographic information?

Did you identify an area where there should be a map?

If yes, where?

Are tables used?

Are they used to show numerical and other relationships?

Did you identify an area where there should be a table?

If yes, where?

[6] As a reminder, using jargon means using words, phrases, or terms that are understood within specific communities but not necessarily understood by someone outside those communities.

Are flowcharts used?

Are they used to show steps in a process?
Are they used to show relationships in a system?
Did you identify an area where there should be a flowchart?

Are organization charts used?

Are they used to show relationships in a hierarchy?
Did you identify an area where there should be an organizational chart?
If yes, where?

Are symbols/icons used?

Are they used to supplement or replace words?
Did you identify an area where there should be a symbol or icon?
If yes, where?

Are screenshots used?

Are they used to show actual physical images of a computer program?
Did you identify an area where there should be a screenshot?
If yes, where?

Are other images used?

What kind?
What is it used for?

Above is the list of common image types as discussed in Gerald J. Alred, Charles Birusaw, and Walter E. Oliu's (2006) book *Handbook of Technical Writing* (pp. 234-245). The descriptions used are from the book, but they are very accurate in describing when various images should be used. When images are used well, they help make a piece of documentation easier to understand. When used poorly, though, they can further complicate the writing.

Are images labeled?

Always Usually Sometimes Rarely Never

Are images relevant to document?

Always Usually Sometimes Rarely Never

Are images explained in the text?

Always Usually Sometimes Rarely Never

Are images near what they explain (Within two paragraphs)?

Always Usually Sometimes Rarely Never

Are images readable (Font, size, colors, etc)?

Always Usually Sometimes Rarely Never

Are images appropriate to the information?

Always Usually Sometimes Rarely Never

Many experts in technical writing have said that images make documentation clearer. If that is the case, then, in theory, documentation that employs visual aids such as pictures, graphs, diagrams, flowcharts, or screenshots will be easier to understand than those that do not. On the other hand, it is possible to use images incorrectly. This question addressed if images, when used, were in appropriate locations near relevant text, were labeled, and were explained. Images can make documentation more accessible because frequently they can replace large amounts of text. However, if the images are inappropriate or used inappropriately, they can hinder accessibility. For example, Desmond D'Souza and Alan Wills (1998) in *Objects, Components, and Frameworks with UML: The Catalysis Approach* clearly say, "Diagrams should be used as part of a narrative explanation and not just on their own" (p. 188).

Is similar information grouped together (is information "chunked")?

Always Usually Sometimes Rarely Never

How? (circle all that apply)

Consistent but distinct use of font or color to group info
Bulleted lists
Numbered lists
Headings and subheading
Chapter divisions
Boxed information or sections

Are bulleted lists used?

Yes/No

To group lists of similar items?

Always Usually Sometimes Rarely Never

Were there places where a bulleted list should have been used but wasn't?

Yes/No

Are numbered lists used?

Yes/No

To list sequential of steps/instructions?

Always Usually Sometimes Rarely Never

To rank the importance of items?

Always Usually Sometimes Rarely Never

Were there places where a numbered list should have been used but wasn't?

Yes/No

Are Concepts Mentioned Before Details?

Always Usually Sometimes Rarely Never

Are Warnings Clearly Identifiable?

Always Usually Sometimes Rarely Never

Do Warnings Appear Before Related Instructions?

Always Usually Sometimes Rarely Never

This question area had several smaller aspects to it. First, it looked at whether the information within the documentation is in a logical order with similar information grouped together (chunking). Additionally, technical writing experts suggest using bulleted lists for non-ordered information and numbered lists for ordered steps. Each item of these lists should begin with a strong action verb (i.e. click, select, type, etc.).

Perceiving Documentation as Usable

Were the sentences of this documentation easy to read?

Always Usually Sometimes Rarely Never

Did the logic of this documentation make sense?

Always Usually Sometimes Rarely Never

Did the organization of this documentation make sense?

Always Usually Sometimes Rarely Never

Were you frustrated by this document?

Yes/No

While I was unable to conduct a usability study given the scope of this project, the usability of documentation is the fourth aspect of access, so is important to touch on. I did so using the above four subjective questions, as each one explored a

user's perceptions that could influence the usability. Research by Aaron Allen, Jinjuan Feng, and Jonathan Lazar (2006) tells us that users are likely to abandon reading documentation they find frustrating and less likely to use the product (p. 150). Their perceptions of ease of reading, sensibility and logic, also affect their frustration level and willingness to stick with the documentation and attempt to use the product. While these are by no means direct measures of usability, they could play a role and results will simply be approached as suggestive.

Readability, logic, and organization are large topics that include many of the best practices mentioned in the "Understanding Documentation" section.

Modifying Documentation

Does the Documentation Discuss Modification?
Yes/No

Because open source software is intended to be modifiable by users, the documentation should include information to assist uninformed users in making modifications. Open source software documentation that does not mention modification is not allowing users of less technical knowledge the same access to the software as those who already know or can figure out how to modify it without documentation.

This is not to imply that the software documentation should teach the users how to program. This question asked if source code modification is mentioned in the documentation.

With more resources, I would look more deeply into modification documentation as well as user documentation. Due to time limitations, however, I focused mainly on user documentation. I asked about modification documentation to gather basic information that can be elaborated on in the future.

3.4: Limitations

There were several other questions that would have been advantageous to ask but were beyond the scope of this study. One issue that was mentioned by critics to open source software documentation is that the documentation is not up-to-date with the current version of the software. Unfortunately, it is frequently very difficult to determine whether the software and the documentation are current with each other. Therefore, this issue was not included in this study.

Additionally, the best way to determine the quality of documentation is to conduct usability tests. Unfortunately, usability tests for documentation and software are very time-

and personnel-intensive and beyond my capability within this study. I did ask my coders' opinions on the readability, logic, organization, and frustration of the documentation. These were factors that are related to perception of usability, but I did not conduct any usability studies.

4: Results and Discussion

4.1: Finding Documentation

Earlier I argued that open source software might discriminate against users if it makes it difficult for the users to find, open, understand or use the documentation. The first question that I attempted to answer was "Can the user find the documentation for these open source sites?"

4.1.1: Many Sites Had No Documentation

In my sample of open source websites, I found that many lacked any documentation whatsoever. Although 12 of the 18 sites (67%) had documentation, I could only access the documentation for 11 of the sites (61%) since one had a broken link. Thus, 7 sites (39%) did not provide any documentation. One can only speculate as to why the sites were missing documentation, but one possibility is that the Open Source Initiative (OSI) does not require documentation, and therefore the developers assumed it was unimportant.

- OSI does not have standards for documentation as they do for software
- OSI only mentions documentation when suggesting developers sell it
- OSI does not require the inclusion of documentation
- OSI does not offer any support for creating documentation as they do for software.

As a result, developers may not have reason to develop documentation.

A second possibility is that developers assume a very narrow, expert audience which understands the software and only intends to use it for purposes identified by the developers. This assumption about audience would be quite problematic since the intention of open source is to be available to a wide group of users who are free to take the software and adapt it to their own purposes.

A third possibility is that the software developers do not think that documentation is necessary because they believe their software is intuitive. Edmund Weiss (1995), professor of communications at Fordham University, describes in his article, "The Retreat from Usability: User Documentation in the Post-Usability Era," a recent trend to make software more intuitive and user-friendly. He explains, however, that "*a friendly* interface is really a *stubborn* interface: a short, inflexible menu of choices, often leading to another and another. From this perspective, menus do not give choices; they limit them. Pull-down menus do not just give the few available options; they 'gray-out' the options that are invalid or inappropriate" (p. 6). Since intuitive software simply means software with limited choices, even intuitive software requires documentation for the features that are included. To omit documentation because of an assumption that the software is intuitive enough to be usable

without written guidance, the software is discriminating against people who rely on that written help.

While scholars such as Jack Herrington and Richard Stallman talk about the poor quality of documentation for open source software, few researchers have explored the extent to which open source software provides any documentation at all.

Since my study focused on documentation specifically, I could conduct no further analysis on those 7 sites that had no documentation; thus, my sampling for the remaining coding was reduced to 11 sites.

4.1.2: Documentation That Existed Was Not Hard To Find, but Additional Web Tools Would Have Made Finding Documentation Easier

With 11 sites remaining, I was able to determine if users could find the documentation within the website. First, I looked at "click depth" which is the count of how many clicks it takes to go from the homepage to the documentation. Web designer Thomas Powell (2000), as mentioned in chapter 2, recommends a click depth of three or fewer (p. 111). The number of clicks for the websites included in my investigation were favorable. All 11 sites had a click depth of 3 or less. 2 of the 11 sites (18%) had a click depth of zero, meaning the documentation was written on the homepage of the website. For all of the websites, the documentation was easily accessible by being within three clicks of the homepage.

Armed with the knowledge that the documentation exists and was not buried too deeply in the website to find, I looked at where on the web pages the documentation was located. 6 of the 11 (55%) had links on the left of the page. 3 of the 11 (27%) had links on the top of the page. 1 of the 11 (9%) had the link in center of the page. 2 of the 11 (18%) did not have links because the documentation was written on the first page. Over half of the websites followed traditional web design practices in link placement as described by human-computer interaction professors, John D. McCarthy, Jens Riegelsberger, and M. Angela Sasse (2005), "Most users expected the navigation menu to be found on the left of the screen" (p. 2). Also, psychologist Michael Bernhard (n.d.) writes, "Internal web links were expected to be located on the upper left side of the browser window." By following standard practices, the developers make the documentation more accessible by allowing users to find it more effectively.

Finally, I looked to see if the people who assembled the web pages had included additional tools such as search bars or site maps to assist users in finding the documentation. I found that 2 of the 11 (18%) had site maps, 4 of the 11 (36%) had search bars, and 1 of the 11 (9%) had both a site map and search bar. In total, 5 of the 11 sites (45%) used at least one additional tool to make the documentation more accessible.

There is good web design, with click depth fitting with the standard of 0-3 clicks and link location fitting with the standard by having links on the left. The organization of the website, however, could be improved if developers included tools such as site maps or search bars. These tools are not mandatory but make the documentation more accessible.

4.2: Opening Documentation

The second aspect of access is opening documentation. This aspect usually refers to technological issues limiting access, for example, whether the documentation was in a format that only some computers could access. Fortunately, all 11 sites with documentation could be opened from any computer with internet access and a PDF reader. The possible exception was that one site used Flash for their documentation. Flash is an external program that must be installed on a computer, but is freely available on the internet for all computers.

4.3: Documentation Formats and Genres

A discussion of the formats and genres of documentation does not fit directly into the four aspects of access (finding, opening, understanding, and using), but is still very important for fully understanding the documentation for open source software. This section determines if the documentation used formats and genres that are static or dynamic. If documentation is dynamic, a reader can contribute to or modify it. Open source software is dynamic in that it is intended for users to be able to access and change the source code, so the documentation should be just as dynamic. If not, then the software is not fully documented, making it is harder for users to access the documentation for modified and rereleased versions.

This section requires a few definitions before moving forward:

- Format: the form in which the documentation is stored and opened by the computer
- Genre: the type of text in which the documentation is written and read by the user
- Static documentation: users can read but not modify the documentation
- Dynamic documentation: users can read *and* modify the documentation

Static documentation is the documentation that most people are accustomed to; examples include instruction books and how-to guides that a user can read to try to find out how to use something or to answer questions. In most cases historically, static documentation has been perfectly sufficient as the products being documented were not dynamic. The development team created a product, the writing team wrote the documentation, and they were released together. With the advent of open source, however, the line between developer and user has been blurred in allowing users to modify the product, so the line between writer and reader should be similarly blurred and allow the readers to modify the documentation. Below, Table 3 shows examples of formats and genres that are static and dynamic.

	Static (non-modifiable)	Dynamic (modifiable)
Format	Traditional website, PDF, .doc, etc	Web 2.0, live chat, helpline, etc
Genre	Manual, FAQ, video	Wiki, forum, blog

Table 3: Examples of Static versus Dynamic Formats and Genres

In short, the goal of this section was to determine whether open source software and its documentation shared the same goals of dynamism or if open source documentation was discriminatory by barring people who modify the software from modifying the documentation. Open source software is designed with a feedback loop giving users the opportunities and tools to modify and redistribute the software. This section examines whether the documentation includes the same feedback loop allowing for modification or if it relies on the writer-driven techniques of traditional proprietary software.

The coding here indicated that both the format and genre documentation for open source software tended to be static and did not usually comply with the ideals of the dynamic field of open source. Table 4 below offers a summary of the formats and genres of the documentation used.

Name	**OFF System**	**netrek**	**more. groupware**	**Logz podcast CMS**	**GlobeCom Jukebox**	**the Noble Ape Simulation**	**CLOCC - Common Lisp Open Code Collection**	**GExperts**	**JXplorer - A Java Ldap Browser**	**Hebrew LaTeX**	**Docutils: Documentation Utilities**	
Target Audience	Computer Industry	Specialized Community (gamers)	Computer Industry	Specialized Community (Podcast Producers)	Mass	Specialized Community (Zoologists)	Computer Industry	Computer Industry	Computer Industry	Computer Industry	Computer Industry	***Percent Used:***
Static Format												
Web 1.0	•	•	•	•	•	•	•	•		•	•	***90%***
PDF									•			***10%***
Flash Guide		•										***10%***
Dynamic Format												
Web 2.0	•		•		•							***30%***
Static Genre												
Manual/How To	•	•		•	•	•	•	•	•		•	***80%***
FAQ	•		•					•	•		•	***50%***
Dynamic Genre												
Forum			•		•							***20%***
Wiki	Broken											***10%***

Table 4: Documentation Formats and Genres of Studied Websites

Figures 3 and 4 are graphs comparing the different formats and genres used. In both, static documentation is in blue and dynamic documentation is in orange. Static documentation was much more common than dynamic documentation, which seems to be in opposition to the goals of open source software, a very dynamic field. The formats of the documentation were very strongly static. 9 of the 11 sites[7] (82%) used traditional websites for their documentation, 1 of the 11 sites (9%) had a downloadable .pdf, and 1 of the 11 sites (9%) had an interactive flash guide. A flash guide gives a sense of user involvement, but since users do not get to modify the documentation, it is still static. For dynamic documentation, 2 of the 11 sites (18%) used web 2.0 (websites that users can write information on as well as read it). In total, less than 20% of websites used dynamic formats.

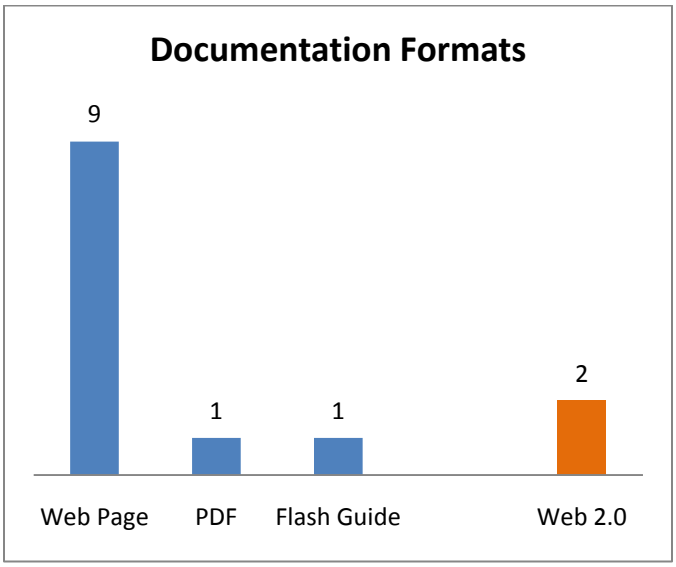

Figure 3: Graph of Documentation Formats

The data for the genre of the documentation showed similar results with much more static documentation than dynamic. 9 of the 11 sites (82%) used a traditional "how-to manual" for their documentation. 5 of the 11 (45%) used an FAQ. FAQs, like the Flash guides, give the impression that they answer users' questions and are dynamic, but the users cannot add questions to the list. In fact, sometimes the FAQ lists are not based on user questions at all and are instead based only on the writers' opinions of what users might ask (Noeldner, 2007). There were three instances of dynamic genres of documentation. 2 of the 11 (18%) used a forum where users could ask and answer questions. Of those 2, 1 was converted to a wiki between my coding of the websites in November 2009 and writing the results in October 2010, and 1 forum is still in use; the last question was asked in July 2010,

[7] An important note: some sites had more than one type of documentation, so the number of documentation types is greater than the total number of sites. Figure 6 above shows the breakdown of which sites used which formats and genres of documentation and, in particular, which formats and genres that are used are static versus dynamic.

although the last answer to a question was posted in January 2010. This long gap in answering questions indicates that whoever is responsible for answering questions, be it moderator or community, is not keeping up to date with helping other users to access the software. Also, as of when coding was done, 1 of the 11 sites (9%) used a wiki, but the wiki contained mostly broken links. In total, only 3 of the 11 sites (27%) used a dynamic genre for documenting software.

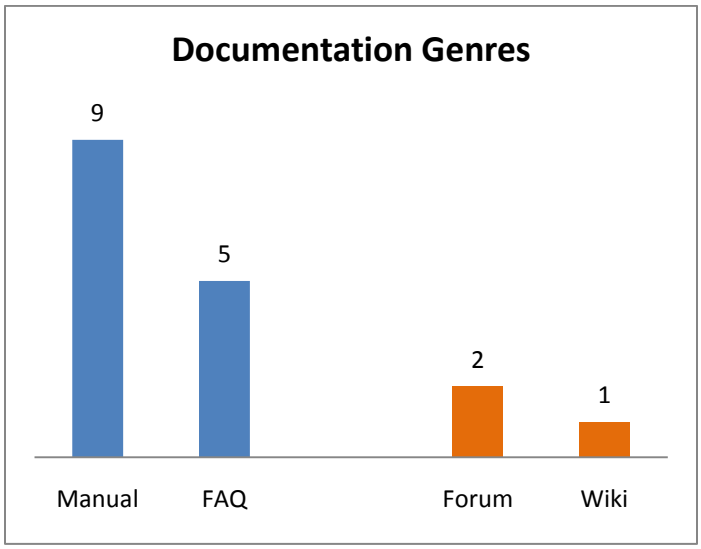

Figure 4: Graph of Documentation Genres

4.3.1: Open Source Documentation Is Static in a Dynamic Field

My research shows that open source documentation is usually static in a dynamic field. This discrepancy could cause serious problems for open source software. For example, if someone modifies a piece of software and redistributes it but cannot modify the documentation, the new developer must choose whether to rewrite all of the documentation or to use the existing, but no longer accurate, documentation. Richard Stallman (2010) of the FSF identifies this as a problem:

> There is a particular reason why the freedom to modify is crucial for documentation for free software. When people exercise their right to modify the software, and add or change its features, if they are conscientious they will change the manual too of— so they can provide accurate and usable documentation with the modified program.

Here, Stallman spells out the importance of having documentation that is as dynamic as the software it is documenting. When the software is changed but the documentation is not, the discrepancy makes it more challenging for users to understand the software and further limits access.

The OSI, however, makes virtually no reference to documentation at all. One of the few places that the OSI website mentions documentation is in its FAQ where it suggests that users sell their documentation as a way to earn money (OSI, n.d.). As I mentioned at length in chapters two and three, the FSF and the OSI have very different goals; the FSF wants open

source software to be idyllic and democratic while the OSI wants it to be an effective financial tool. OSI, however, includes "non-discrimination" in its definition, but does not address the fact that dynamic software without dynamic documentation limits access when the software has been modified. My research supports Stallman's assertions of the importance of dynamic documentation yet reflects how the OSI does not value such documentation.

4.3.2: Traditional Websites as a Common Middle Ground of Accessibility

Another observation from this data is that the common use of static websites indicates that the websites were designed with a mass audience in mind instead of the technical audience that many scholars assume writers of technical documentation consider. 9 of the 11 sites (82%) use traditional static websites as the format for providing documentation to users. If the developer was interested only in simplicity of development, then word documents or PDFs would have been the simplest. If the developers were offering the best way for users to interact with the data, they would have used Web 2.0. Static websites were a middle ground between ease of writing and ease of using. Since the OSI and FSF both mention that open source is primarily an online community, using online documentation was the logical choice.

This data seems to run counter to Jack Herrington (2003)'s assertion that only developers can use OS documentation. He asks, "Is [the documentation] only intended for engineers already using the tool?" The answer: no, because if it was, it would be in a format that only engineers could read. Anyone can open a website. Websites allow *universal access*. The people who write open source documentation are accused of not considering audience when writing, but this act shows that there is some consideration of audience.

The frequent use of traditional web pages improves access and lessens discrimination by allowing anyone who can access the internet to see the documentation. Websites written in HTML, the standard programming language for websites, can also be read by accessibility tools such as translators and text readers that further expand the audience. The use of static instead of dynamic documentation, however, limits access by requiring that modified software has entirely new documentation in order to be up to date. This involves a lot of work that some developers may choose not to put in, resulting in incomplete or obsolete documentation that is difficult to read and understand, especially by people who most need the documentation.

4.4: Understanding Documentation

A great deal of research has been conducted on how to write clear, understandable instructional materials. The literature identifies many "best practices" involving the use of jargon[8], proper English, images, and organization. In this part of my analysis I looked at

[8] As a reminder, using jargon means using words, phrases, or terms that are understood within specific communities but not necessarily understood by someone outside those communities.

these issues as the ones most people think of when talking about the shortcomings of open source software documentation. These shortcomings make it difficult for people who do not already understand the software to use it.

4.4.1: Writers of Documentation Relied Heavily on Jargon

Analysis of the frequent use of jargon on these websites indicated that the people writing the documentation usually assume narrow rather than mass audiences. They assume their audience is already very computer-literate and able to understand complicated computer terms. In addition, they may be addressing a special-interest group related to the content of the software by using jargon that was likely to only be known by that specific group.

To create context for the jargon, my coders determined the target audience of the software based on descriptions on the website. The results showed that the computer community and industry was the most common audience while software targeted for a "mass audience" was extremely uncommon. Only 1 of the 11 sites with documentation (9%) targeted a mass audience. 3 of the 11 websites (27%) targeted a specialized community not within the computer industry. Specifically, the sites in this study targeted video gamers, podcast producers, and zoologists. 7 of the 11 sites viewed (64%) targeted an audience in the computer industry.

Knowing the audience the software targets allowed my coders and me to look at the jargon used in the documentation and to see if the writers attempted to expand the audience by defining jargon or reducing it. Unfortunately, I was forced to omit one of the sites from the study of jargon as it was written entirely in Hebrew with no translation. This is an accessibility issue in itself, but not one covered by the scope of this project. Of the sites written in English, 9 of the 10 (90%) used terms that my coders and I determined to be jargon, while 1 of the 10 sites (10%) did not use jargon. Interestingly, the one site that did not use jargon was not the site targeting a mass audience.

Jargon is only discriminatory against people unfamiliar with it if it is undefined. Jargon with definitions educates the users instead of excluding them. There are many ways of defining jargon such as using a glossary, parenthetical definitions, footnotes, and hyperlinks. Of the 9 sites which used jargon, however, 5 (56%) failed to define it.

Of the remaining 4 sites that did define their jargon, 3 of them (75% of defined, 33% of total with jargon) used parenthetical definitions, although 1 did only rarely. The other two used parenthetical definitions frequently but not for all instances of jargon. 2 sites sometimes used hyperlinks to define jargon, but 1 of those had broken hyperlinks impeding definition. 1 website had a glossary. Footnotes, margins, and in-text definitions were never used. Coders determined that only 1 of the 9 websites (11%) used definitions that made the terms easier to understand. There is no correct way to define jargon, as long as it is defined. These statistics, however indicate the typical ways that the jargon is defined

The sites used jargon equally for software-related concepts and industry ideas. The sites were divided into thirds with regard to whether the jargon used referred to the software, the community or industry, or both. The jargon was identified as software-specific in 3 of

the 9 sites (33%), as industry or community-specific in 3 of the 9 sites (33%), and as both community and software specific in 3 of the 9 sites (33%). There seemed to be a reliance on jargon for both community-specific and software technical terms. Both types of jargon are discriminatory if undefined, but they give some insight into the assumed audience of the people writing the documentation. This study suggested that the assumed reader was a member of the targeted community as well as very familiar with computers.

The 9 sites using jargon generally used it very frequently per page. Sample pages from 5 of the websites (56%) had over 21 words or terms that were considered to be jargon while 4 sites (44%) had fewer than 20 terms. Very frequent use of jargon is particularly important when jargon is undefined. Perhaps the frequency of jargon explains why jargon was so rarely defined; the people writing the documentation underestimated how much of their language was, in fact, jargon and used it more frequently than they were defining it. The writers may not have realized that they were using more than 20 technical terms on over half of the websites.

A possible explanation for the ubiquity of jargon is that the writers of the documentation assumed the audience already understood what the jargon meant and that it, in fact, was not jargon. Robert Bly (n.d.) a scholar of technical writing, explains on his website, "Technical writers sometimes prefer to use big, important-sounding words instead of short, simple words. Technical terms are helpful shorthand when you're communicating within the profession, but they may confuse readers who do not have your special background." The writers were a part of their own target audience so assumed that readers had the same knowledge that they did. With open source software, though, this simply is not the case. Users do not fit neatly into the presumed audiences of the writers because some of the users want to modify the software to use for a different purpose. Software whose documentation employs jargon discriminates against people who are not within the assumed audience.

4.4.2: Sample Documentation Followed the Best Practices for Documenting Proprietary Software, but these Procedures Were Not Always Executed Effectively

Another observation from the questions about "understanding documentation" was that documentation tended to include the features of the technical documentation genre as defined for proprietary software, but they did not implement them effectively or consistently. This was true with use of proper English, images, and organization. There is no defined genre yet for open source software documentation, but by looking at the protocols for proprietary software documentation I was able to get a general idea of the quality of the documentation.

It seems that open source documentation is written in the genre of other technical writing that takes an engineering approach to software documentation and does not exist in its own genre. While people writing the documentation for the open source software appear to be familiar with most of the protocols for technical writing, the finer nuances tend to get lost.

In a time when many professional technical writing positions are being outsourced offshore to cut costs, it is reassuring to find that the people writing the documentation for the

open source software I examined have a firm grasp of how to write in proper English. The exception to this was the site that was written in Hebrew, but since it was clearly targeting other Hebrew-speakers and as I do not read Hebrew, I could not judge if it used proper grammar or spelling.

Proper English is a very simple metric to determine how easy or hard it is to understand a document. Frequent spelling and grammar errors, the document is harder to read. Additionally, technical writing is a specific genre of writing and, therefore, has its own protocols and expectations. Table 5 below outlines the best practices that the websites followed.

	Percentage of Websites Usually or Always Following Best Practice
Used Culture and Gender Neutral Language	100%
Written in the 2nd Person	90%
Written in the Present Tense	80%
Used Proper English Grammar and Spelling	70%
Began Sentences with Command Verbs	20%

Table 5: Websites Following Best Practices in Writing

The simplest and most obvious practices were followed very frequently. Writers, in general, know that documentation should be in the present tense, second person, and not use language that obviously alienates anyone. The finer details, however, such as beginning sentences with command verbs (i.e. *click* the button, *enter* your name, etc.) are less intuitive.

I drew a similar conclusion when I examined the documentation for organization. Coders reported that most of the documents were at least somewhat organized using "chunking," grouping information with similar information to make it easier to find. Of the 10 sites written in English, 6 (60%) were always "chunked," 3 (30%) were usually chunked, and 1 (10%) was sometimes chunked. None of the websites were rarely or never chunked indicating that the people writing the documentation knew to put like information together.

There are several different ways to indicate that information is grouped together. The most common was headings, with coders reporting that 7 (70%) of website used this method. 5 (50%) of websites used chapter divisions according to the coders it. Additionally, they reported 2 (20%) of websites used visual cues (font, color, lines, whitespace, etc) and 1 (10%) used sections that do not have headings. All of these are acceptable ways of telling the readers that they have left one 'chunk' of information and have moved on to another.

Another important aspect of proper organization is writing the general concepts before explaining the details. To explain the details first would be equivalent to giving someone an untranslated copy of Homer's Iliad before teaching him Greek. It is ineffectual and frustrating to the reader. This may seem intuitive, but the results show that "general concepts first" was not implemented as frequently as it should have been. Only 1 site out of 10 (10%) consistently had large concepts before details, according to the coders. They

reported that 3 sites (30%) usually did, 4 sites (40%) sometimes did, 1 (10%) did rarely, and 1 site (10%) never had concepts before details.

Parallel structure, using the same organization in all sections across the paper, is also useful for readability and to increase understanding. None of the websites always used parallel structure, but 6 (60%) usually did, 1 (10%) sometimes did, and 3 (30%) rarely used parallel structure.

These factors of writing and organization show that the people writing the technical documentation for the open source software had some understanding of the best practices used in technical writing. The documentation fit with the protocols of the genre of technical documentation for software and, therefore, did not decrease access in confusing the readers in what they are looking at. The implementation of these best practices could be improved to make documentation easier to read and understand, thus increasing access. Documentation should be better organized by implementing "chunking" and parallel structure more consistently, as well as making sure that details are not covered until after concepts. These small changes make documentation easier for all readers to access by removing some of the confusion that had been blocking understanding.

4.4.3: Images Are Used but Not Used Effectively

Most technical writing manuals recommend using images to supplement writing. 9 of the 11 sites examined by coders (82%) used at least one image in the document. Screenshots were the most common, utilized by 7 of the 9 sites with images (77%). Of those 7 sites, 6 (86%) used screenshots properly to show actual images of the computer software. Additionally, 4 of the 9 sites with images (44%) used tables, 2 of the 9 sites (22%) used symbols or icons, and 1 of the sites (11%) used a flow chart. Drawings, maps, and organizational charts were never used on the websites examined.

But simply using images is not sufficient; websites need to use images effectively. My coders judged the effectiveness of the use of images in the documentation. Irrelevant images are one possible problem that makes use of visuals in documentation less effective. Fortunately, relevance was rarely a problem. Of the 9 sites, 8 (89%) were deemed to always use relevant images while only 1 (11%) did not. Additionally 7 of the 9 sites (78%) always placed their images near relevant text while 1 (11%) did only sometimes and 1 (11%) never did.

Another concern with using images is that they need to be properly labeled in order to explain how they are related to the text. An image without labels may be visually appealing, but it is unlikely to be informative and to make the documentation more accessible. Of the 9 sites that my coders and I looked at, 4 (44%) always used labeled images, 2 (22%) usually did, 1 (11%) sometimes did, and 2 (22%) never labeled their visuals.

Finally, like text, images need to be displayed in such a way that it is easy to read. If an image is too small, blurry, or too crowded, it is not of value. For these 9 sites, 2 (22%) always had readable images, 4 (44%) usually did, 2 (22%) sometimes did, and 1 (11%) never had images that were easy to read.

Figure 5 below gives an example of an image that my coders and I agreed did not do anything correctly. The image appeared to be a diagram showing the software's relationship to other items in a network. There was minimal information, however, to clarify or explain this image. It had a webpage to itself with no text except for a title, "The Vision", and the jargon-filled, barely-intelligible caption, "DMCA Compliant Attornet [sic] Resistant Redundant Distributed Storage with Anonymous Universal Access In The Cloud." The page[9] was linked from the toolbar at the side of every page, so there was no linking webpage to provide context. As for the image itself, the text in the image was not always large enough to be readable, and there were many parts not labeled at all, yet were vague enough that they need labels in order to be understood. This was considered the worst image that was found, but several other images shared at least one trait with it.

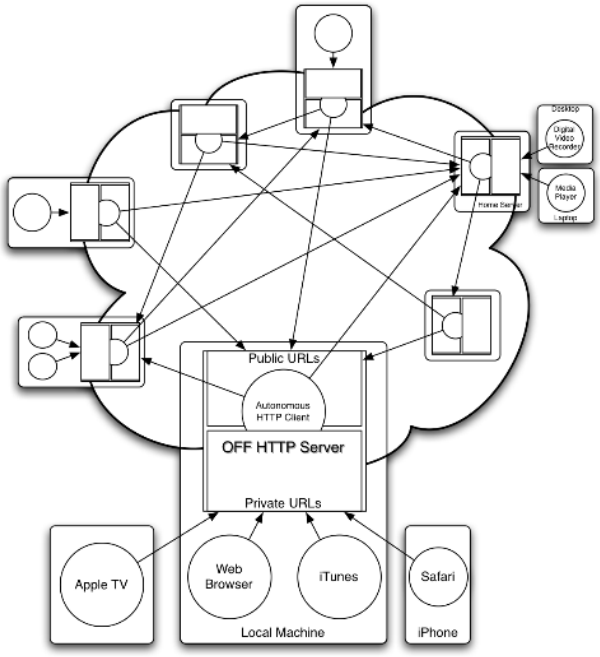

Figure 5: A Poorly Implemented Visual--OFF System, 2008

Images are supposed to be used in documentation to supplement the text by providing a point of reference. As the old adage states, "a picture is worth a thousand words," so by using images in documentation, the writer or developer is able to make complex concepts much simpler to understand without having to verbally describe them. This is only true, however, for images that are used effectively. There is no benefit to access or understanding if the user cannot read a visual or understand why the visual was used.

Screenshots are the most common type of visual, and there are several explanations for this. One is that screenshots are very effective as a point of reference when telling someone how to use a piece of software. Instead of having to describe a menu or toolbar, a screenshot shows it and helps the user understand the context of instructions.

[9] http://offsystem.sourceforge.net/why-off/, 2008

Another explanation is that screenshots are very fast, cheap, and easy images to provide. They do not require a professional artist or special software. All current computers have a key or set of keys to press in order to take a screenshot and automatically save it in an easy-to-use format. It takes no more time or effort, and, in fact, often less, to create a screenshot than it does to write about it.

In spite of the benefits of screenshots, there are places and times that other types of visual are more appropriate. For example, computer software usually requires or creates a process, and some do both. In spite of this, only 1 website used a flowchart which is the correct type of visual for showing a process. Similarly, icons, such as those found on buttons and in menus, are growing to have universal meanings that could help a user better understand a document, yet only 2 of the 9 (22%) chose to use these.

While the people writing the documentation for this sampling of open source software used images, they were not used effectively nor often enough to benefit the users by increasing access. My coders were required to read or search large amounts of text without the benefit of helpful images as references.

4.5: Using Documentation

After determining whether users could find, open, modify, and understand the documentation for open source software, the only remaining question was whether or not they could use it. I did not study usability per se but asked my coders whether they perceived the documentation as sensible and easy to read, factors that may play a role in usability. I asked my coders four questions soliciting their opinions of the documentation overall:

- Were the sentences of this documentation easy to read?
- Did the logic of this documentation make sense?
- Did the organization of this documentation make sense?
- Were you frustrated by this document?

Up to this point, the assumption was that following the best practices of technical writing creates the most accessible documentation for open source software. While scholars may deem that to be the case, the true test is if users find the documentation easy enough to understand that they can use it to increase their access to the software. I analyzed my coders' responses to the four questions noted above to ascertain if a relationship existed between professionally-defined best practices and reader-determined usability.

4.5.1: Overall, Readers Find Most Documentation Is Readable, Logical, and Organized

While I did not conduct any usability tests on the documentation, the readers' opinions on the readability, logic, and organization represented their perceptions influencing

usability. The results in Figure 6 below show that, for these three factors, more than half of the documents were usually or always perceived positively by my coders.

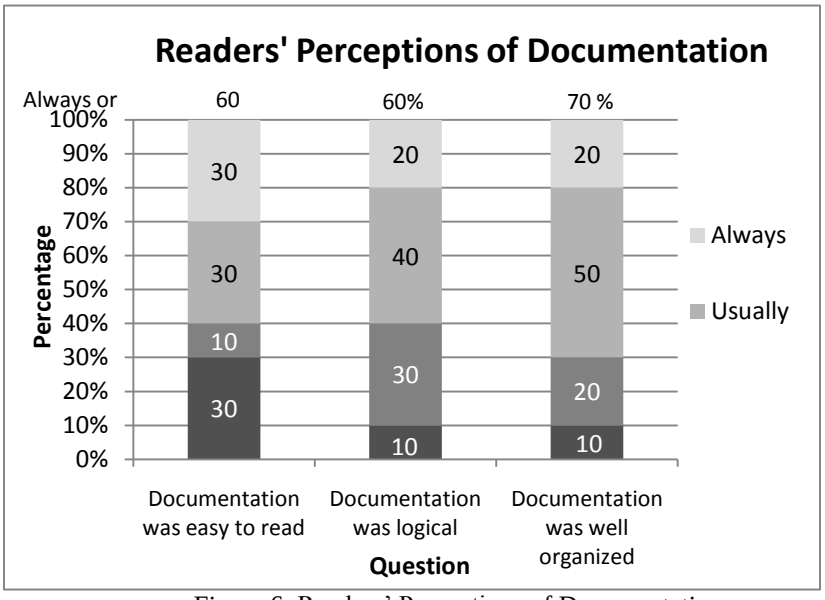

Figure 6: Readers' Perceptions of Documentation.

When asked about the ease of reading the documentation, raters said that 3 of the 10 documents (30%) were rarely easy to read, 1 (10%) was sometimes easy to read, 3 (30%) were usually easy to read, and 3 (30%) were always easy to read. In summary, 60% were generally readable while 40% were not. Fortunately, none of the sites were judged by readers as being never readable.

The results for the question, "Did the logic of this documentation make sense?" were very similar to the previous question about readability. Raters said that 1 of the 10 documents (10%) was rarely logical, 3 (30%) was sometimes logical, 4 (40%) were usually logical, and 2 (20%) were always logical. In summary, once again, 60% generally were logical while 40% were not. None of the sites were judged by readers as being never logical.

The third question, "Did the organization of this document make sense?" had very similar responses as well. Coders said that 1 of the 10 documents (10%) was rarely well organized, 2 (20%) were sometimes well organized, 5 (50%) were usually well organized, and 2 (20%) were always well organized. In general, 70% of the documents were well organized and 30% were not. The increase from 60% in the past two questions to 70% in this question is not statistically significant as a difference of 10% accounts for only one site.

Across the 10 websites, 9 (90%) were rated either rated positively for all questions or negatively for all questions. Only one site was ranked positively for one question and negatively for the other two. This accounts for the difference between 60% positive response for the first two questions and 70% positive response for the last question.

4.5.2: Users Found Documentation Frustrating Over Half the Time

The fourth and final question I asked in the usability section was, "did you find the documentation frustrating?" This question may seem vague, but to me it is the most indicative of the usability of the documentation. The results showed that of the 10 documents written in English, my coders found 6 (60%) to be frustrating and 4 (40%) not frustrating. Aaron Allen, Jinjuan Feng, and Jonathan Lazar (2006), computer science and human-computer interaction researchers, have studied user frustration extensively. They have concluded that when users are frustrated by technology, they are less productive and efficient. They write in their article for the journal *Interacting with Computers* that in extreme cases, frustration can lessen productivity by as much as 50% (p. 150). The documentation may follow all of the rules, protocols, and expectations of technical writing, but if a user finds it frustrating, it will be less effective. More importantly, if some users find it frustrating while others do not, the documentation is less accessible by the users who are frustrated as they have a harder time using it.

That might be a bit of an overstatement as documentation that follows all existing protocols for technical writing is unlikely to be frustrating, but there were 2 of the 10 websites in this study that were considered "always" usable for readability, logic, and organization, but were still seen as frustrating to the users. I clarified these two responses with the coder who said that one was frustrating because of a lack of information and the other was frustrating because of a glut of information which made it challenging to find any one specific piece. Both were well written, organized, and logical, yet still frustrating due to the inability of the reader to use the documentation to answer specific questions.

4.6: Modifying Documentation

I asked one more question that does not seem to fit into any of the above categories, but is very important when examining anything relating to open source software. I asked whether the documentation made any mention of modifying the software. This mention could take many forms, from detailed instructions for how to modify the software to simply a mention of how to find the source code. Any mention would show that the writers of the documentation are aware that open source software is designed for modification, but that some of the users may not inherently have the necessary knowledge or skills. This is an extra level of accessibility that could be added. Of the 10 sites written in English, 4 of them (40%) mentioned modification. Fewer than half chose to add this information to help people modify the software regardless of their technical knowledge or previous experience.

5: Conclusion

I conducted this study to examine how and to what extent open source software is discriminatory. The open source definition requires all software using an open source license to "not discriminate against any person or group of people" (OSI, n.d.), and in this study I sought to discover how that applied to the documentation of the software, both theoretically and practically. The particular group that I hypothesized was being discriminated against was people of limited technical or industry specific knowledge and experience.

I chose eighteen pieces of open source software on sourceforge.net that 4 co-coders and I analyzed both quantitatively and qualitatively for features that might limit access in four aspects: finding the documentation, opening it, understanding it, perceiving it as being usable, and being able to modify it.

My first discovery was that a large number of the websites I examined - more than one third - had no documentation. These websites clearly limited access to documentation and, therefore, to the software, by failing to include documentation for a general audience unfamiliar with the software. This is discrimination because people who have technical or specific community knowledge may be able to use the software without explanation, but there is no documentation to assist those without that knowledge

There are several possible explanations for why so much documentation did not exist. It is possible that the software developers think their software is intuitive enough that it does not need documentation. It is also possible that, since the OSI does not require documentation for open source software, developers are simply choosing not to include it. A third possibility is that the software developers assume their target audience already knows enough about the software, and that people without that existing knowledge would not be likely to use the program.

Other findings from this study suggest the last might be true. 90% of the documentation written in English used words or phrases that only the specialized target audience would understand, indicating the writers of this documentation were not concerned about reaching general audiences.

Not all of my findings showed that the documentation was discriminatory. People writing this documentation understand some of the best practices of the technical writing genre and attempt to include them (e.g., writing in the second person, using the present tense, etc.). Most also made the documentation easy to find, using acceptable website navigation principles. In nearly all cases in my study, the documentation was written using standard English spelling and grammar. A few of the finer nuances of web design and some of the best practices were missing but, on the whole, most of the writers had a firm grasp of the genre and were ability to apply the practices in an accessible way.

A common shortcoming was the use of images. Nearly all of the websites employed images, but such usage was far from ideal. Images were sporadic and frequently unlabeled, illegible, or otherwise distracting instead of helpful.

As seen in images and writing, documentation writers seemed familiar with the use of best practices, but often fell short with their implementation. The documentation did not always use command verbs and rarely included effective use of images. There are a few possible explanations for this. First, the developers or writers may not be trained in how to write documentation. Some of these people may be amateurs creating software in their free time, while others might be developers at small companies without a trained technical writer. Another possible reason is that creating proper documentation takes a large amount of resources: time, labor, or money. Startup companies and amateurs do not have the resources to spend on careful documentation.

The aspect of the documentation examined in this study that, in my opinion, may cause the most discrimination was the continued use of traditional, static documentation in a field as dynamic as open source software. As a field, open source software has blurred the line between user and developer so that anyone who wants to change a piece of existing open source software is not only allowed but encouraged to do so. It seems counterproductive, then, for most open software documentation to be written in a way that cannot be modified along with the software. Additionally, static documentation is one directional. It assumes and constrains a narrow audience. The biggest issue that this causes is that modified or redistributed versions of the software are unlikely to have up-to-date documentation because of the effort required to create it.

There are a few possible reasons why the documentation for open source software remains static even in a dynamic field. First, static documentation is the most familiar and the has the most writing about how to effectively create it. Only recently have developers started using dynamic media and formats for user documentation. People write what they know, and instruction manuals written on static web pages are known by most people. Another possible explanation is that dynamic documentation requires more resources to develop and maintain. It is easy to write text and put it on a website or in a PDF. It takes effort to find or create a dynamic form and seed it with information. It takes even more effort to moderate the website constantly to make sure questions are being answered and that everyone is using the website for appropriate, relevant purposes. With static documentation, the developer only needs to put time in once.

This study was a cursory overview of the flaws with the documentation of open source software. It opens the door for several future potential studies. The first would be to do this or a similar study on a larger scale. While I selected the eighteen websites for this study based on very specific and well-thought out criteria, the study was still very limited in scope. A survey of more websites using similar criteria would gather data that is more statistically robust due to a larger base size.

A second potential study would be to look at documentation of open source software with respect to time to determine if there is a trend related to the effectiveness or discrimination in design of the documentation. My study only looked at a single point in time, but it would be useful to learn if the documentation on a set of sites improves over time or if the documentation shifts and improves overall as open source software becomes more

common. This could be done by looking at the documentation of several specific pieces of software and noting if they change over time.

The third is a usability test for open source documentation. I made assumptions about usability, but only based on the opinions of a small number of people and on the best practices for technical writing as defined by professionals. Whether someone believes documentation to be easy to read and follow and whether it is actually usable are two different issues. Proper usability tests would greatly benefit the field in order to determine the effectiveness of the documentation.

The fourth is a study to see if the documentation makes it easier for someone to modify the software. I briefly touched on this by counting how many of the websites mentioned modification, but I did not go into any detail in my research. The documentation should not need to teach the user how to program, but it should offer advice and have well documented, non-obfuscated code that a potential modifier could easily navigate. In addition to research to determine if the documentation improves access for using the software, usability tests for the documentation with regards to modification would truly be a test to see if the documentation is keeping up with open source software.

The fifth potential study is to investigate the relationship between user-reported frustration and a piece of documentation's use of professionally-determined best practices. This project did not include enough websites to be able to statistically determine if using the best practices for technical writing is correlated with user frustration, but a larger study could provide this valuable data.

In summary, my study suggests that open source software may, in fact, limit access and discriminate by omitting documentation, by not applying the best practices of technical writing, by using static documentation to represent dynamic software, and by failing to provide even basic assistance with modifying the software. First, people who require documentation are often discriminated against by there not being any documentation for software. Documentation that does not exist cannot be accessed. Second, people outside of the narrow audience that the developers assume are using the software cannot understand the documentation because of the heavy reliance on jargon, thus limiting their access. Third, by using static documentation in a dynamic field, people who modify the software are less able to change the documentation, leading to obsolete documentation for revised versions of the software. Finally, for many reasons including some outside the scope of this project, readers' access is limited by their frustration with the documentation.

These conclusions lead to several suggestions. Listed below is advice for open source documentation writers based on my research:

- **Include documentation for your software**. Failing to have documentation discriminates against people who need instruction to use the software, even if it is only to answer a question.

- **Do not assume your user knows about your software or the community the software is intended for.** It is safe to assume the audience is intelligent and interested, but avoid jargon when possible. If it is not possible to avoid jargon, define it so users who are not informed can look up the meanings of the technical terms.
- **Create a dynamic document**. Open source is a dynamic field that invites people to modify the software. Allow them to modify the documentation, too. I recommend a wiki, although a forum can also be effective by allowing users to ask and answer questions. The format and genre do not matter as long as they allow the documentation to be revised.
- **Include Modification Information**. The documentation is not expected to teach users how to program, but it should make it easier for those inclined to modify the software to do so. Mention in the documentation the language used to write the software as well as any advice that would be helpful in modification. Also, make sure the source code for the software is commented well and not obfuscated.
- **Research and use the best practices for technical writing**. Follow their recommendations, because these are the protocols that have been proven to be effective.
- **When writing documentation, consider using info mapping techniques or technical writing guides.** Reference these on your website to help users modify and rewrite the documentation.

An additional piece of advice is for the OSI: If OSI insists that the software be nondiscriminatory, license should require software to include accessible documentation written for non-specialist users, and not just the specialized audience assumed to be most likely to use the software. Then and only then will open source software be truly accessible to everyone.

References

Allen, A., Feng, J., & Lazar, J. (2006). Determining the impact of computer frustration on the mood of blind users browsing the web. *Assets 2006. The Eighth International ACM Sigaccess Conference on Computers & Accessibility, Oct 23-25, 2006, Portland, Oregon, USA* (pp. 149-156). Portland, Oregon: ACM

Alred, G. J., Brusaw, C. T., & Oliu, W. E. (2006). *Handbook of technical writing, Eighth edition.* New York: St. Martin's Press.

Bernhard, M. L., & Larson, L. (n.d.). What is the best layout for multiple-column web pages? *Usability News 3.2*. Retrieved September 15, 2010, from http://wsupsy.psy.twsu.edu/surl/usabilitynews/3S/layout.htm

Blanco, E. (2008, June 9). Documentation issues - OSS Watch Wiki. *OSS Watch Wiki*. Retrieved from http://wiki.oss-watch.ac.uk/DocumentationIssues

Bly, R. W. (n.d.) Improving your technical writing skills. *Bob Bly - Copywriting Services*. Retrieved March 22, 2010, from http://www.bly.com/Pages/documents/File146.htm

D'Souza, D. F., & Wills, A. C. (1998). *Objects, components, and frameworks with UML: The Catalysis approach (Addison-Wesley Object Technology Series)*. New York: Addison-Wesley Professional.

Feng, D. J., Hochheiser, D. H., & Lazar, D. J. (2010). Diaries. *Research Methods in Human-Computer Interaction* (p. 127). New York, NY: Wiley.

Hackos, J. T., & Stevens, D. M. (1997). *Standards for Online Communication*. New York, NY: Wiley.

Hartley, J., & O'Sullivan, T. (1993). Genre. *Key concepts in communication and cultural studies (Studies in Culture and Communication)* (2 ed., p. 128). New York: Routledge.

Herrington, J. (2003, April 15). Is documentation holding open source back?. *DevX*. Retrieved October 13, 2010, from http://www.devx.com/opensource/Article/11839

Horn, R. E. (n.d.) Information Mapping - 40 years of information mapping. *Information Mapping - About Us*. Retrieved September 15, 2009, from http://www.infomap.com/index.cfm/AboutUs/40_Years_of_Information_Mapping

Horton, S., & Lynch, P. J. (2009). *Web style guide: Basic design principles for creating web sites, 3rd edition.* New Haven: Yale University Press.

IEEE (2001). *IEEE 1063: IEEE standard for software user documentation*. Institute of Electrical & Electronics Engineers.

Jansen, C., & Steehouder, M. (1994). *Quality Of technical documentation.(Utrecht Studies in Language and Communication 3)*. Atlanta, GA: Editions Rodopi.

Katz, S. B. (1992). The Ethic of expediency: Classical rhetoric, technology, and the Holocaust. *College English, 54*(3), 255-270.

McCarthy, J. D., Riegelsberger, J., & Sasse, M. A. (2005). Commercial uses of eyetracking. *Human-Computer Interactions, 8*, 1-2.

McMurrey, D. A. (2001). *Power tools for technical communication --2001 publication.* Boston, MA: Heinle, 2001.

NIST Biometric Image Software (NBIS). (2006, November 30). *National Institute of Standards and Technology*. Retrieved from http://fingerprint.nist.gov/NFIS/

Nagle, R. (n.d.) Idiotprogrammer Article: Does open source documentation suck? *Idiotprogrammer.* Retrieved March 2, 2009, from http://www.idiotprogrammer.com/computers/linuxdoc.php

Noeldner, C. (2007, June 9). WinWriters - FAQs on the web. *WritersUA - Home page*. Retrieved from http://www.writersua.com/0011cn.htm

Open Source Initiative (n.d.). *Open Source Initiative*. Retrieved March 25, 2008, from http://opensource.org/

Powell, T. A. (2000). *Web design: The complete reference* (1st ed.). New York: Mcgraw-Hill Companies.

Raymond, E. S. (2001). *The cathedral & the bazaar: Musings on Linux and open source by an accidental revolutionary*. Sebastopol: O'Reilly Media, Inc.

Sommerville, I. (2001, July 1). Software documentation. *Literate programming*. Retrieved from www.literateprogramming.com/documentation.pdf

SourceForge.net: Download and Develop Open Source Software for Free (n.d.). *SourceForge*. Retrieved October 10, 2009, from http://www.sourceforge.net

Stallman, R. (n.d.). The Lifelong Activist. *Richard Stallman's Personal Page*. Retrieved October 23, 2010, from http://stallman.org

Stallman, R. (2001, May 29). Free software: Freedom and cooperation. Speech presented from New York University, New York.

Stallman, R. (2010, January 21). The free software definition. *The GNU operating system*. Retrieved, from http://www.gnu.org/philosophy/free-sw.html

Stallman, R. (2010, July 1). Why free software needs free documentation - GNU Project - Free Software Foundation (FSF). *The GNU Operating System*. Retrieved from http://www.gnu.org/philosophy/free-doc.html

Sun, H. (2006). The triumph of users: Achieving cultural usability goals with user localization. *Technical Communication Quarterly, 15*(4), 457-481.

The OFF System --The Vision. (2008, August 25). *The OFF System -- OFF System Introduction*. Retrieved from http://offsystem.sourceforge.net/why-off/

Vanderheiden, G. C. (1996, May 6). Universal design. What it is and what it isn't. *Trace Research and Development Center - Trace Center*. Retrieved from http://trace.wisc.edu/docs/whats_ud/whats_ud.htm

Walsh, I. (n.d.). User Guides Tips: Making difficult subjects easy to understand. *Business, proposal, technical writings tips and templates - Klariti Business tips for smart people*. Retrieved March 3, 2009, from http://www.klariti.com/technical-writing/How-to-Improve-User-Guides.shtml

Weber, S. (2004). *The success of open source*. Cambridge: Harvard University Press.

Weiss, E. (1995). The retreat from usability: User documentation in the post-usability era. *SIGDOC Asterisk Journal of Computer Documentation*, *1337*, 3-18.

Wheeler, D. A. (2007, April 16). Why open source software / free software (OSS/FS, FOSS, or FLOSS)? Look at the numbers!. *David A. Wheeler's personal home page*. Retrieved, from http://www.dwheeler.com/oss_fs_why.html

Young, M. (2002). *Technical writer's handbook: Writing with style and clarity* (New Edition ed.). Sausalito, CA: University Science Books.

Appendix

Appendix A: Software Used

Below is a chart indicating the name, genre, ranking, and a short description of the eighteen programs I selected to study, sorted by genre. The descriptions are provided by the software to sourceforge.net for use on the site

Genre	Name	Rank	Website	Description
Communications	OFF System	1276	offsystem.sourceforge.net	The OFF System for content storage and retrieval, lets you store all digital content and allows only the people authorized to use it to do so. It is the proof of concept for 'bright nets' and will allow anyone to securely share digital data legally.
Communications	customer Connect	22105	customerconnect.org	customerConnect is a customer service software support solution comprising live interactive customer chat (interAct), ERMS (Email Response Management System) (emailGateway), and an online help center (knowledgeBase).
Games/ Entertainment	Labyrinth of Worlds	10869	low.sourceforge.net	LoW is a rewrite of the first-person role-playing game Ultima Underworld II: Labyrinth of Worlds that came out in the early 1990s. One of the most celebrated game of its genre, this rewrite attemps to recapture the minutiae and spirit of the original.
Games/ Entertainment	netrek	24666	netrek.org	Netrek is a multiplayer battle simulation game with a Star Trek theme. Up to 16 players are divided into two teams that fight each other for dominion over the galaxy.
Internet	more. groupware	2361	moregroupware.de	Web-based groupware written in PHP. Including modules like webmail, notes, todo, contacts, project management, calendar and others.
Internet	Logz podcast CMS	8152	logz.org	The purpose of the "Logz" (podcast) is to offer a whole freedom using (multimedia / sound / animation) while taking advantage of dynamic management of the data (CMS) and gives you a full ablility to do flash, ascii or html layouts.
Multimedia	Guitar Scales	15063	guitarscales.sourceforge.net	Welcome,This project is all about the guitar! It's a Java based program that shows you all kind of scales on a guitar arm. It's really usefull for learning, developing your skills or writing solo's.
Multimedia	GlobeCom Jukebox	23269	gjukebox.sourceforge.net/	Powerful and reliable mp3 jukebox with web based interface. Key featuresinclude sophisticated random song selection, web based interface, integratedripper, streaming, multi-user capable, album cover art.

Office/Business	Market Analysis System	14980	eiffel-mas. sourceforge.net	System for analysis of financial markets using technical analysis. Includes facilities for stock charting and futures charting, as well as automated generation of trading signals based on user-selected criteria. Operates on both daily and intraday data.
Office/Business	Versatile Maintenance Tracker	8947	vmt.sourceforge.net	VMT (formerly Vehicle Maintenance Tracker) tracks the maintenance of multiple properties. Property can include vehicles, boats, planes, buildings, etc. This project is comparible to Auto-Do-It. Since this program uses Java, it is cross-platform.
Scientific/ Engineering	Discontinuous Deformation Analysis	24270	dda.sourceforge.net	Discontinuous Deformation Analysis is discrete element method useful for simulating the motion of large numbers of individual bodies in independent motion,subject to contact constraints.
Scientific/ Engineering	the Noble Ape Simulation	13038	nobleape.com/sim/	Simulates a biologically diverse tropical island, and the ape inhabitants cognitive processes. For MacOS Classic and X, with Java, Windows and Linux(Motif) versions in beta. Features a non-polygonal graphics engine (Ocelot) and a command-line version.
Software Development	CLOCC - Common Lisp Open Code Collection	4619	clocc.sourceforge.net	Our aim is to create a collection of useful and free Common Lisp - Applications that are easily portable among the various CL - Implementations.
Software Development	GExperts	3421	gexperts.org	GExperts is a free set of tools built to increase the productivity of Delphi and C++Builder programmers by adding several features to the IDE. GExperts is developed as Open Source software and encourages user contributions to the project.
System	JXplorer - A Java Ldap Browser	5681	jxplorer.org	A free java ldap client with LDIF support, security (inc SSL, SASL & GSSAPI), translated into many languages (inc. Chinese), online help, user forms and many other features.
System	SNEeSe	18015	sneese.sourceforge.net	SNEeSe is an emulator for the Nintendo SNES console for x86 PCs. SNEeSe is written in 32-bit C, C++, and NASM x86 assembly. Project goal is to make as accurate, functional, and usable an emulation core as is reasonably possible.
Text Editors	Hebrew LaTeX	22306	ivritex.sourceforge.net	IvriTeX is a project spunned off heblatex and it's purpose is to maintain the Hebrew LaTeX support, and provide a meeting point for Hebrew TeXers for the coordination of improving the Hebrew support.
Text Editors	Docutils: Documentation Utilities	1604	docutils.sourceforge.net	Utilities for general- and special-purpose documentation, including autodocumentation of Python modules. Includes reStructuredText, the easy to read, easy to use, what-you-see-is-what-you-get plaintext markup language.

Appendix B: Final Coding Sheet

Finding and Opening Documentation

Is there user documentation?

No **Yes**

How many clicks does it take to navigate from the home webpage to the documentation?

0-3 ---------4-6 ---------7+

Can documentation be found using a search bar?

No **Yes**

Can documentation be found using a site map?

No **Yes**

Where on the page are links to the documentation?

Place an X in the box where the links are located on the webpage:

What medium/media is/are used for the documentation?

Circle all that apply:

Web page
Web 2.0 (Wiki, Forums, etc.)
PDF
Word document/.doc
Other _____

What genre(s) is/are used for the documentation?

Circle all that apply:

Manual/How-To
FAQ
Forum
Wizard
Searchable database
Video tutorial
Wiki
Blog
Other _____

Does the documentation discuss modification?

No **Yes**

Understanding Documentation

Who do you think is the target audience of this documentation?

Computer industry
Other professional industry
Coding communities
Specialized non-industry community
Mass use
Other specific audience:
_____

Is there jargon (technical terms that might be unfamiliar to some people)?

No **Yes**

On one page, how many *undefined* jargon words/terms are there?

0-10 **11-20** **21+**

Jargon (defined and undefined) is:
Circle all that apply:

Industry/Community related?
Software related?
Other?

_____

Is jargon defined using parenthetical definitions?

Never **Always**
1 2 3 4 5

Is jargon defined in a "definitions" section at the beginning of the document?

Never **Always**
1 2 3 4 5

Is jargon defined in the margins or in footnotes?

Never **Always**
1 2 3 4 5

Is jargon defined using hyperlinks

Never **Always**
1 2 3 4 5

Is jargon defined using the format of "term – definition"?

Never **Always**
1 2 3 4 5

Do the definitions for jargon make the terms easier to understand?

Never **Always**
1 2 3 4 5

Is the documentation written in the 2nd person? (command form or "you")

No **Sometimes** **Yes**

Does the documentation use culture-and-gender-neutral language?

No **Sometimes** **Yes**

Does the documentation use correct grammar, punctuation, and spelling?

No **Sometimes** **Yes**

Does the documentation use the present tense?

No **Sometimes** **Yes**

Does the documentation begin instructions in the command form by starting sentences with an action verb (click, start, listen, etc)?

No **Sometimes** **Yes**

Is there a glossary (chapter at the end of the document for definitions)?

No **Yes**

Understanding Documentation Cont.

Are images used?
No Yes

How many images are on the page?
_____

Are screenshots used?
No Yes
Are they used to show actual physical images of a computer program?
No Yes
Did you identify an area where there should be a screenshot and isn't?
No Yes
If yes, where? _____
_____

Are flowcharts used?
No Yes
Are they used to show steps in a process?
No Yes
Did you identify an area where there should be a flowchart and isn't?
No Yes
If yes, where? _____
_____

Are symbols/icons used?
No Yes
Are they used to supplement or replace words?
No Yes
Did you identify an area where there should be a symbol or icon and isn't?
No Yes
If yes, where? _____
_____

Are drawings used?
No Yes
Are they used to show objects and spatial relationships?
No Yes
Did you identify an area where there should be a drawing and isn't?
No Yes
If yes, where? _____
_____

Are maps used?
No Yes
Are they used to display geographic information?
No Yes
Did you identify an area where there should be a map and isn't?
No Yes
If yes, where? _____
_____

Are tables used?
No Yes
Are they used to show numerical and other relationships?
No Yes
Did you identify an area where there should be a table and isn't?
No Yes
If yes, where? _____
_____

Understanding Documentation Cont.

Are organization charts used?

No Yes

Are they used to show relationships in a hierarchy?

No Yes

Did you identify an area where there should be an organizational chart and isn't?

No Yes

If yes, where? _____
_____

Are other image types used?

No Yes

What kinds? _____
_____

What are they used for? _____
_____

Are images labeled?

Never **Always**

1 2 3 4 5

Are images relevant to document?

Never **Always**

1 2 3 4 5

Are images near relevant text? (Within two paragraphs)

Never **Always**

1 2 3 4 5

Are images easily readable? (Font, size, colors, etc)

Never **Always**

1 2 3 4 5

Understanding Documentation

Is similar information grouped together (is information "chunked")?

Never **Always**

1 2 3 4 5

How?

Circle all that apply:

Consistent but distinct use of font, color, lines, or white space

Headings and subheading

Chapter divisions

Boxed information or sections

Other:

_____

Are concepts mentioned before details?

Never **Always**

1 2 3 4 5

Are warnings used?

No **Yes**

Are warnings clearly identifiable?

Never **Always**

1 2 3 4 5

Do warnings appear before related instructions?

Never **Always**

1 2 3 4 5

Is parallel structure used? (are grammar and form the same in similar items?)

Never **Always**

1 2 3 4 5

Are bulleted lists used?

No **Yes**

Are they used to group lists of similar items?

No **Yes**

Are they used to list sequential steps/instructions?

No **Yes**

Are they used to rank the importance of items?

No **Yes**

Did you identify an area where there should be a bulleted list and isn't?

No **Yes**

If yes, where? _____

_____

Are numbered lists used?

No **Yes**

Are they used to group lists of similar items?

No **Yes**

Are they used to list sequential steps/instructions?

No **Yes**

Are they used to rank the importance of items?

No **Yes**

Did you identify an area where there should be a numbered list and isn't?

No **Yes**

If yes, where? _____

_____

User Perceptions Influencing Using Documentation

Were the sentences easy to read?

Never **Always**

1 2 3 4 5

Did the logic make sense?

Never **Always**

1 2 3 4 5

Did the organization make sense?

Never **Always**

1 2 3 4 5

Were you frustrated by this document?

No **Yes**

Is there anything else you wish to say about the documentation?

_____

_____

_____

_____

_____

_____

_____

_____

Thank You