
COMPUTER ETHICS

OPEN SOURCE SOFTWARE

OLIVER MEZQUITA PRIETO

Index

- 1. Introduction, 1**
- 2. History, 2**
 - 2.1. Unix, 2
 - 2.1.1. Multics, 2
 - 2.1.2. Unix, 3
 - 2.1.3. The Unix Wars, 4
 - 2.2. The GNU Project, 4
 - 2.2.1. MIT, 4
 - 2.2.2. The GNU Project, 5
 - 2.3. Linux, 6
 - 2.3.1. Minix, 6
 - 2.3.2. Linux, 6
- 3. Society and Culture, 8**
 - 3.1. Government, 10
 - 3.2. Ethics, 10
 - 3.3. Media, 10
 - 3.4. Education, 12
 - 3.5. Innovation Communities, 12
 - 3.6. Arts and recreation, 13
- 4. Open Source vs. Closed Source, 14**
 - 4.1. Collaboration and project management, 14
 - 4.2. Commercialization, 15
 - 4.3. End user support, 17
 - 4.4. Innovation, 18
 - 4.5. Compatibility and interoperability, 19
 - 4.6. User freedoms to exploit software, 20
 - 4.7. Integration and overall “feel”, 20
 - 4.8. Security, 22
- 5. Criticism, 23**
- 6. References, 24**

1. Introduction

Open source is a set of principles and practices that promote access to the production and design process for various goods, products, resources and technical conclusions or advice. The term is most commonly applied to the source code of software that is made available to the general public with relaxed or non-existent intellectual property restrictions. This allows users to create user-generated software content through incremental individual effort or through collaboration.

Some consider open source as one of various possible design approaches, while others consider it a critical strategic element of their operations. Before open source became widely adopted, developers and producers used a variety of phrases to describe the concept; the term open source gained popularity with the rise of the Internet and its enabling of diverse production models, communication paths, and interactive communities. Later, open source software became the most prominent face of open source practices.

The open source model of operation can be extended to open source culture in decision making which allows concurrent input of different agendas, approaches and priorities, in contrast with more centralized models of development such as those typically used in commercial companies. "Open source" as applied to culture defines a culture in which collective decisions or fixations are shared during development and made generally available in the public domain- - - as seen with Wikipedia. This collective approach moderates ethical concerns over a "conflict of roles" or conflict of interest. Participants in such a culture are able to modify the collective outcomes and share them with the community.

2. History

The History of Open Source is tied to three operating systems: Unix, GNU, and Linux. These were projects that initially shaped the identity of the open source community, beginning in the 1960s and continuing to the present day, and proved that open source is a viable software development model. No study of open source is complete without understanding the history of these systems.

2.1. Unix

2.1.1. Multics

In the late 1960s, Bell Labs, owned by AT&T, began working collaboratively with General Electric and the Massachusetts Institute of Technology to create a new operating system. The system, known as Multics, was to be used in-house at Bell Labs. Although Multics was a significant achievement in the realm of computer science, it was also time-consuming and expensive. Its goals were too lofty for Bell Labs to achieve, and in 1968 Bell began to withdraw from the project (Hauben 1994).

Some of the last people to work on Multics, at Bell Labs, were Ken Thompson, Dennis Ritchie, and Joe Ossanna. For these three and others, the loss of Multics was a disaster. At the time, other operating systems lacked the flexibility and simplicity that Multics had promised. Thompson and the rest decided to build a new operating system to suit their programming needs. After Bell Labs rejected their requests for a new computer, the group found an obsolete PDP-7 computer on which to begin their efforts. They called the new operating system Unix to distinguish it from Multics and began work.

Multics was distributed in 1975 to 2000 by Groupe Bull in Europe, and in the U.S. by Bull HN Information Systems Inc. In 2006 Bull SAS open sourced Multics versions MR10.2, MR11.0, MR12.0, MR12.1, MR12.2, MR12.3, MR12.4 & MR12.5

2.1.2. Unix

Work progressed smoothly on Unix throughout the early 1970s. The group acquired new computers, developed the high-level programming language C, making Unix portable, and created many new tools to make Unix more useful. Within Bell Labs, other departments began to use Unix for a variety of tasks. Unix eventually became the standard for Bell's computing needs, and a development support group, called Unix Support Group, was formed to provide support for a standard version of Unix.

Even in its infancy, word of Unix was spreading throughout the computing world. Unix was particularly appealing to the academic computer science community. Academic institutions were able to purchase licenses for the Unix source code very cheaply. Government and commercial licenses were much more expensive.

In November of 1973, Professor Bob Fabry of the University of California at Berkeley attended a presentation on Unix at Purdue University. His interest was piqued, and Fabry convinced Berkeley to purchase a PDP-11/45 computer capable of running the current version 4 of Unix. In January of the next year, Version 4 Unix was installed with the help of Ken Thompson of Bell Labs.

In the fall of 1975, Ken Thompson decided to take a one-year sabbatical from Bell Labs to teach at Berkeley, his alma mater. The Computer Science department had just purchased the new PDP-11/70 computer, and Thompson helped to install the latest version of Unix, Version 6, on it. Two graduate students, Bill Joy and Chuck Haley, also arrived in 1975, and began working on a Pascal compiler written by Thompson, a new text editor, and improvements to underlying parts of the Unix system itself.

Other programmers began to take interest in the new Pascal compiler at Berkeley, and during the year of 1977, Joy began to distribute the "Berkeley Software Distribution," an open source version of Unix containing the improvements and additions made at Berkeley. BSD was sold for a nominal fee to people who had already obtained a Unix license from AT&T. In mid-1978, Joy put together the "Second Berkeley Software Distribution," or 2BSD, which was distributed the following year. Distribution jumped from about thirty copies with the original BSD to about seventy-five copies with 2BSD.

Improvements continued, increasing portability, improving memory usage, and implementing new tools. In December 1979, 3BSD was released, and nearly 100 copies were shipped. At this time, with the breakup of Bell, the price of Unix licenses for the academic community began to increase. AT&T shifted management of Unix to a new group and began emphasizing proprietary versions of Unix (Dibona et al.: McKusick 1999). The first of these new releases was System III in 1982, followed in 1983 by System V. Berkeley moved to fill the void Bell had left in open distributions and continued to release further versions of BSD, using a new open source software license known as the BSD License.

AT&T did not begin heavy commercial promotion of Unix until the mid-1980s. What they found when they ventured into the market was that many vendors were already selling their

own proprietary version of Unix. The issue of the day was which version of Unix would become dominant. In 1987, in an effort to unify the market, AT&T formed an alliance with Sun Microsystems, a strong supporter of BSD. In response to the move, several vendors created the Open Software Foundation (OSF) to support their own open source versions of Unix. AT&T and Sun in turn formed Unix International. Thus the "Unix Wars" began (The Open Group 2001).

2.1.3. The Unix Wars

Throughout the remainder of the 1980s and into the 1990s, the Unix Wars raged. During this time, many different versions of Unix were released, both proprietary and open source. In 1991, AT&T spun off Unix System Laboratories, which passed through several hands before being bought in 1995 by Santa Cruz Operation (SCO).

Three distinct versions of BSD emerged from the Unix Wars: FreeBSD, known for its simplicity, stability, and ease of use; NetBSD, known for its portability and research-oriented environment; and OpenBSD, known for its high level of security and stability (Howard). Other versions of open source Unix are available as well, including a version from SCO itself and Darwin, the foundation on which Apple's Mac OS X is built.

2.2. The GNU Project

2.2.1. MIT

In 1971, Richard Stallman, a Harvard undergraduate student, began working at the MIT Artificial Intelligence Lab, primarily on the ITS, an operating system unique to the computers at MIT. The community at the MIT AI Lab was a small group of programmers who improved code by passing it back and forth; in other words, the group's software development basis was open source (Rasch 2000).

In the early 1980s, the community at MIT AI Lab began to collapse, due in part to computer architecture advances that rendered ITS obsolete. Computers that were replacing MIT's PDP-10s had their own operating systems, but none were open source. Even getting an executable copy meant signing a non-disclosure agreement.

At the same time that ITS became obsolete, the AI Lab community also disbanded. One of the first people to move away was Brian Reed of Carnegie Mellon University. Instead of sharing his text-formatting program Scribe, with the AI Lab community, he sold it to a commercial company, which altered the code to insure profits rather than communal improvements. Soon, spin-off companies began breaking away from the community. Eventually, nearly all of the programmers were hired away to work on commercial software projects. Richard Stallman was left with a choice:

"One: join the proprietary software world, sign the non disclosure agreements and promise not to help his fellow hackers. Two: leave the computer field altogether. Or three, look for a way that a programmer could do something for the good. He asked himself, was there a program or programs he could write, so as to make a community possible again?" (Rasch 2000).

2.2.2. The GNU Project

Stallman's ideals of software were lofty: he wanted free software for the masses. According to Stallman, the definition of a truly free software is a program that allows users the right to run the program for any purpose, modify the program to suit their needs, redistribute copies with open source, and distribute modified versions of the program with open source (Stallman 2002).

Stallman decided to start creating open software by developing an operating system, the most crucial software for using a computer. He anticipated a "community of cooperating hackers" that would develop around the project much as in the MIT AI Lab (Stallman 2002). He chose to make the operating system Unix compatible because that was the dominant system at the time. Stallman picked the acronym GNU for his project, according to a hacker custom of creating recursive acronyms. GNU stands for "GNU's Not Unix" (Stallman 2002).

Stallman resigned from MIT in January 1984 so that MIT would have no claim on distributing GNU. He would avoid GNU becoming proprietary software at all costs. However, he was invited by the head of the MIT AI Lab to continue using the MIT facilities.

In 1985, Stallman founded the Free Software Foundation (FSF), a tax-exempt charitable organization, to support the free software development being done on the GNU Project. Stallman also contributed to the project by writing a multiple language compiler known as GCC, a debugger (GDB), a text editor (GNU Emacs), and other software.

In order to ensure that GNU would remain open source in future, Stallman created the GNU General Public License (GPL). The GPL specified "that users of the source code could view, change, or add to the code, provided they made their changes available under the same license as the original code" (GNU General Public License). Stallman received the MacArthur fellowship, which entails a stipend of \$500,000, in 1990 for his work with GNU, the GPL, and the FSF.

The GNU operating system continued to grow throughout the 1990s, developing piece by piece. Each piece was implemented on a Unix system, so that components could be completed and distributed before the entire system was released. By 1990, the only major piece missing from the system was the kernel. A kernel is the core of an operating system "that provides basic services for all other parts of the operating system" ("Kernel").

Stallman's team began work on a kernel in 1990, called Hurd. However, work is progressing slowly.

2.3. Linux

2.3.1. Minix

In 1987, a professor, Andrew Tanenbaum, invented Minix, a clone of the Unix operating system to be used for educational purposes. Minix was not the most sophisticated of operating systems, but its appeal to programmers worldwide was that all 12,000 lines of C and assembly were available to be studied and tinkered with (Hasan 1999).

2.3.2. Linux

In August of 1991, Linus Torvalds, a 21 year old Computer Science student at the University of Helsinki, posted to the Minix users newsgroup that he was working on a new, free operating system, adding parenthetically that it was "just a hobby, won't be big and professional like GNU" (Newitz). What Torvalds was actually creating was a kernel, the core of an operating system.

In 1992, the completed Linux kernel was combined with the incomplete GNU operating system, resulting in a working open source operating system. According to Stallman, "It is due to Linux that we can actually run a version of the GNU system today" (Stallman 2002). In later years, this combination of GNU and Linux along with other free software exploded in popularity and became commonly known simply as Linux.

In September, version 0.01 of the Linux kernel was released on the net, and enthusiasm began to rise around the project (Hasan 1999). On October 5th, Torvalds sent a formal call for volunteers to the Minix newsgroup, saying "Are you without a nice project and just dying to cut your teeth on a OS you can try to modify for your needs?" (Newitz 2000). Torvalds' plea appealed to many programmers' senses of curiosity and excitement. By December, version 0.10 was released, still as a bare-bones kernel.

And soon after version 0.12 was released, Linux was then licensed under the GNU General Public License to ensure that the source would be free to all (in fact the readme for that release stated that it will soon be licensed under that license. Linus was just checking for licensing issues before doing so). In the ensuing years, thousands of people began working with Linux, helping to improve the kernel itself or writing software for use on Linux systems.

Throughout the 1990s, as Linux swelled in popularity and became more and more sophisticated, vendors began distributing it commercially. Although Linux was and is free

and open source, vendors such as Red Hat, Novell and Mandriva have gathered it into a format that more closely resembles other contemporary operating systems. With the addition of graphical user interfaces and other user-friendly features, these distributors were able to profit from selling open source bundled into a product that everyday users wanted and could use with ease (Newitz 2000).

3. Society and culture

Open source culture refers to creative practices that involve the appropriation and/or free sharing of found or created content. Examples of open source culture include collage, found footage film, music, and appropriation art. Open source as applied to culture defines a culture in which fixations are made generally available. Participants in such an open source culture are able to modify those products, if needed, and redistribute them back into the community or other organizations.

The rise of open-source culture in the twentieth century resulted from a growing tension between creative practices that involve appropriation, and therefore require access to content that is often copyrighted, and increasingly restrictive intellectual property laws and policies governing access to copyrighted content. The two main ways in which intellectual property laws became more restrictive in the twentieth century were extensions to the term of copyright (particularly in the United States) and penalties, such as those articulated in the Digital Millennium Copyright Act (DMCA), placed on attempts to circumvent anti-piracy technologies.

Although artistic appropriation is often permitted under fair use doctrines, the complexity and ambiguity of these doctrines creates an atmosphere of uncertainty among cultural practitioners. In addition, the protective actions of copyright owners create what some call a "chilling effect" among cultural practitioners.

In the late twentieth century, cultural practitioners began to adopt the intellectual property licensing techniques of free software and open-source software to make their work more freely available to others, including the Creative Commons.

The idea of an "open source" culture runs parallel to "Free Culture," but is substantively different. Free Culture is a term derived from the free software movement, and in contrast to that vision of culture, proponents of OSC maintain that some intellectual property law needs to exist to protect cultural producers. Yet they propose a more nuanced position than corporations have traditionally sought. Instead of seeing intellectual property law as an expression of instrumental rules intended to uphold either natural rights or desirable

outcomes, an argument for OSC takes into account diverse goods (as in "the Good life") and ends.

One way in achieving the goal of making the fixations of cultural work generally available is to maximally utilize technology and digital media. As predicted by Moore's law, the cost of digital media and storage plummeted in the late 20th Century. Consequently, the marginal cost of digitally duplicating anything capable of being transmitted via digital media dropped to near zero. Combined with an explosive growth in personal computer and technology ownership, the result is an increase in general population's access to digital media. This phenomenon facilitated growth in open source culture because it allowed for rapid and inexpensive duplication and distribution of culture. Where the access to the majority of culture produced prior to the advent of digital media was limited by other constraints of proprietary and potentially "open" mediums, digital media is the latest technology with the potential to increase access to cultural products. Artists and users who choose to distribute their work digitally face none of the physical limitations that traditional cultural producers have been typically faced with. Accordingly, the audience of an open source culture faces little physical cost in acquiring digital media.

Essentially born out of a desire for increased general access to digital media, the Internet is open source culture's most valuable asset. It is questionable whether or not the goals of an open source culture could be achieved without the internet. The global network not only fosters an environment where culture can be generally accessible, but also allows for easy and inexpensive redistribution of culture back into various communities. Some reasons for this are as follows.

First, the internet allows even greater access to inexpensive digital media and storage. Instead of users being limited to their own facilities and resources, they are granted access to a vast network of facilities and resources, some for free. Sites such as Archive.org offer up free web space for anyone willing to license their work under the Creative Commons license. The resulting cultural product is then available to download for free (generally accessible) to anyone with an internet connection.

Second, users are granted unprecedented access to each other. Older analog technologies such as the telephone or television have limitations on the kind of interaction users can have. In the case of television there is little, if any interaction between users participating on the network. And in the case of the telephone, users rarely interact with any more than a couple of their known peers. On the internet, however, users have the potential to access and meet millions of their peers. This aspect of the internet facilitates the modification of culture as users are able to collaborate and communicate with each other across international and cultural boundaries. The speed in which digital media travels on the internet in turn facilitates the redistribution of culture.

Through various technologies such as peer-to-peer networks and blogs, cultural producers can take advantage of vast social networks in order to distribute their products. As opposed to traditional media distribution, redistributing digital media on the internet can be virtually costless. Technologies such as BitTorrent and Gnutella take advantage of various

characteristics of the internet protocol (TCP/IP) in an attempt to totally decentralize file distribution.

3.1. Government

Open source government – primarily refers to use of open source software technologies in traditional government organizations and government operations such as voting.

Open politics (sometimes known as Open source politics) – is a term used to describe a political process that uses Internet technologies such as blogs, email and polling to provide for a rapid feedback mechanism between political organizations and their supporters. There is also an alternative conception of the term Open source politics which relates to the development of public policy under a set of rules and processes similar to the Open Source Software movement.

Open source governance – is similar to open source politics, but it applies more to the democratic process and promotes the freedom of information.

3.2. Ethics

Open Source ethics is split into two strands:

Open Source Ethics as an Ethical School - Charles Ess and David Berry are researching whether ethics can learn anything from an open source approach. Ess famously even defined the AoIR Research Guidelines as an example of open source ethics.

Open Source Ethics as a Professional Body of Rules - This is based principally on the computer ethics school, studying the questions of ethics and professionalism in the computer industry in general and software development in particular.

3.3. Media

Open source journalism – referred to the standard journalistic techniques of news gathering and fact checking, and reflected a similar term that was in use from 1992 in military intelligence circles, open source intelligence. It is now commonly used to describe forms of innovative publishing of online journalism, rather than the sourcing of news stories by a professional journalist. In the Dec 25, 2006 issue of TIME magazine this is referred to as user created content and listed alongside more traditional open source projects such as Linux.

Weblogs, or blogs, are another significant platform for open source culture. Blogs consist of periodic, reverse chronologically ordered posts, using a technology that makes webpages easily updatable with no understanding of design, code, or file transfer required. While corporations, political campaigns and other formal institutions have begun using these tools

to distribute information, many blogs are used by individuals for personal expression, political organizing, and socializing. Some, such as LiveJournal or WordPress, utilize open source software that is open to the public and can be modified by users to fit their own tastes. Whether the code is open or not, this format represents a nimble tool for people to borrow and re-present culture; whereas traditional websites made the illegal reproduction of culture difficult to regulate, the mutability of blogs makes "open sourcing" even more uncontrollable since it allows a larger portion of the population to replicate material more quickly in the public sphere.

Messageboards are another platform for open source culture. Messageboards (also known as discussion boards or forums), are places online where people with similar interests can congregate and post messages for the community to read and respond to. Messageboards sometimes have moderators who enforce community standards of etiquette such as banning users who are spammers. Other common board features are private messages (where users can send messages to one another) as well as chat (a way to have a real time conversation online) and image uploading. Some messageboards use phpBB, which is a free open source package. Where blogs are more about individual expression and tend to revolve around their authors, messageboards are about creating a conversation amongst its users where information can be shared freely and quickly. Messageboards are a way to remove intermediaries from everyday life - for instance, instead of relying on commercials and other forms of advertising, one can ask other users for frank reviews of a product, movie or CD. By removing the cultural middlemen, messageboards help speed the flow of information and exchange of ideas.

OpenDocument is an open document file format for saving and exchanging editable office documents such as text documents (including memos, reports, and books), spreadsheets, charts, and presentations. Organizations and individuals that store their data in an open format such as OpenDocument avoid being locked in to a single software vendor, leaving them free to switch software if their current vendor goes out of business, raises their prices, changes their software, or changes their licensing terms to something less favorable.

Open source movie production is either an open call system in which a changing crew and cast collaborate in movie production, a system in which the end result is made available for re-use by others or in which exclusively open source products are used in the production. The 2006 movie *Elephants Dream* is said to be the "world's first open movie", created entirely using open source technology.

An open source documentary film has a production process allowing the open contributions of archival material, footage, and other filmic elements, both in unedited and edited form. By doing so, on-line contributors become part of the process of creating the film, helping to influence the editorial and visual material to be used in the documentary, as well as its thematic development. The first open source documentary film, "*The American Revolution*," which will examine the role that WBCN-FM in Boston played in the cultural, social and political changes locally and nationally from 1968 to 1974, is currently in production by the production company, Lichtenstein Creative Media.

Open Source Filmmaking refers to a form of filmmaking that takes a method of idea formation from open source software, but in this case the 'source' for a film maker is raw unedited footage rather than programming code. It can also refer to a method of filmmaking where the process of creation is 'open' i.e. a disparate group of contributors, at different times contribute to the final piece.

Open-IPTV is IPTV that is not limited to one recording studio, production studio, or cast. Open-IPTV uses the internet or other means to pool efforts and resources together to create an online community that all contributes to a show.

3.4. Education

Within the academic community, there is discussion about expanding what could be called the "intellectual commons" (analogous to the creative commons). Proponents of this view have hailed the Connexions Project at Rice University, OpenCourseWare project at MIT, Eugene Thacker's article on "Open Source DNA", the "Open Source Cultural Database", openwebschool, and Wikipedia as examples of applying open source outside the realm of computer software.

Open source curricula are instructional resources whose digital source can be freely used, distributed and modified.

Another strand to the academic community is in the area of research. Many funded research projects produce software as part of their work. There is an increasing interest in making the outputs of such projects available under an open source license. In the UK the Joint Information Systems Committee (JISC) has developed a policy on open source software. JISC also funds a development service called OSS Watch which acts as an advisory service for higher and further education institutions wishing to use, contribute to and develop open source software.

3.5. Innovation communities

The principle of sharing predates the open source movement; for example, the free sharing of information has been institutionalized in the scientific enterprise since at least the 19th century. Open source principles have always been part of the scientific community. The sociologist Robert K. Merton described the four basic elements of the community - universalism (an international perspective), communism (sharing information), disinterestedness (removing one's personal views from the scientific inquiry) and organized skepticism (requirements of proof and review) that accurately describe the scientific community today. These principles are, in part, complemented by US law's focus on protecting expression and method but not the ideas themselves. There is also a tradition of publishing research results to the scientific community instead of keeping all such knowledge proprietary. One of the recent initiatives in scientific publishing has been open

access - the idea that research should be published in such a way that it is free and available to the public. There are currently many open access journals where the information is available for free online, however most journals do charge a fee (either to users or libraries for access). The Budapest Open Access Initiative is an international effort with the goal of making all research articles available for free on the internet. The National Institutes of Health has recently proposed a policy on "Enhanced Public Access to NIH Research Information." This policy would provide a free, searchable resource of NIH-funded results to the public and with other international repositories six months after its initial publication. The NIH's move is an important one because there is significant amount of public funding in scientific research. Many of the questions have yet to be answered - the balancing of profit vs. public access, and ensuring that desirable standards and incentives do not diminish with a shift to open access.

Benjamin Franklin was an early contributor eventually donating all his inventions including the Franklin stove, bifocals and the lightning rod to the public domain after successfully profiting off their sales and patents.

New NGO communities are starting to use the open source technology as a tool. One example is the Open Source Youth Network started in 2007 in Lisboa by ISCA members.

3.6. Arts and recreation

Copyright protection is used in the performing arts and even in athletic activities. Groups have attempted to protect such practices from being fettered by copyright

4. Open Source vs. Closed Source

4.1. Collaboration and project management

Closed source projects ("CS") tend to collaborate either only to a limited or peripheral degree with third parties (other than project co-members), or under non-disclosure agreements. Corporate development is usually run by teams or structured groups, with workload, agendas, intended results, and deadlines, centrally agreed, and use paid developers to achieve these goals as required.

Open source projects ("OS") by contrast usually embrace third party involvement with enthusiasm. Project involvement tends to be voluntary for many of those involved, and harnesses the enthusiasm of participants who are usually given in return, freedom to do what they feel best suited to, and allowed to become involved, committed, or (in many cases) learn as they do so. New code is developed and reviewed in a less formal process by many people – in some cases hundreds of thousands, or millions – and since many of these are intimately familiar with the system concerned, the quality of review and final writing tends to be extremely high on such projects and the speed of development can be very fast. Collaborative work is also resource efficient, since duplication of effort is avoided.

CS and OS projects tend to view their priorities as different. CS projects tend to work to deadlines, the date at which (for corporate, market or investor purposes) a new product or update must be released, or a new feature made available. OS projects, lacking investor pressure, tend to be more actively interested in how to do a job well, as well as produce it quickly, and in producing work to a high standard.

Example:

- Microsoft Windows is closed source, it is the world's most common operating system and a de facto standard in the computer world. Proponents would tend to agree that almost every version of Windows has been a major leap from previous versions, in practically every area, both in innovation, and ease of use. Because it is developed by

one company, it can be centrally managed and co-ordinated, and there are fewer "odd gaps" in its development as such. However against this, a huge effort has gone into protecting and preventing others from benefiting from this work in unintended ways, there has been much conflict over "hidden code" allegations, and security and quality have consistently been criticized by many third parties over the years.

- Wikipedia is open source, both its software (mediawiki) and the actual content, is collaborative. In five years it grew from under 1000 articles, to millions of articles, and its software is continually updated by a developer community that spans the world. The software is as a result very robust, since millions of people have access to it and any untoward happening has been analyzed by many developers at a moment's notice. The information contained is broader, more comprehensive and more in-depth than any corporate team could produce, and grows extremely fast in quality and scope, there is no censorship or hidden agendas, and there is a huge user base of millions of contributors, however as there is no central control, there are many articles not yet up to the intended long term standard and no article can be 100% relied upon without additional checking by the user.

4.2. Commercialization

The primary mechanisms for making money from closed-source software all seem to involve imposition of artificial scarcity constraints on something that, by its nature, can be very easily and cheaply copied and distributed. It has famously been said that "information wants to be free"; closed-source vendors would counter this by saying that "information providers want to be paid". Thus, they impose various limitations on what can be done with their software, first of all by usually not giving customers access to the source code, and then backing this up by restrictions on copying, enforced using both legal (copyright law) and technological measures (copy protection and digital rights management).

Thus, in closed-source software, there is an element of the design which means that the product is designed to prevent the customer from doing some actions that the company feels would result in compromise of their source code, even if this is something that isn't source compromising, or something the consumer wishes to do.

Open-source, on the other hand, abandons all such attempts at forcing the customer to do things in a certain way. Instead, the revenue model is based solely on what customers can be persuaded to pay for of their own free will.

Another important factor in the closed-source revenue model involves fending off competitors (both actual and potential) by continually raising the barriers to entry. Thus, new versions of the software are continually being introduced, with lots of new features being added. Competitors then have to come up with their own answers to these new features (otherwise they will not be seen to "remain competitive"), which they have to reinvent essentially from scratch, which adds to their own costs. Typically these features are

added with little thought for their impact on the conceptual integrity of the overall product, leading to the well-known phenomenon of software bloat.

Another problem with the addition of these features is that they often add to the software vendor's own costs; when they try to adapt the product to new markets and new applications, then the more feature-ridden the product is, the less flexible and adaptable it becomes. For example, operating systems built on the Linux kernel are available for a wider range of processor architectures than Microsoft Windows, including PowerPC and SPARC. None of these can match the sheer popularity of the x86 architecture, nevertheless they do have significant numbers of users; yet Windows remains unavailable for these alternative architectures, because the cost of porting it would be far too great.

The most obvious complaint against open source software revolves around the fact that making money through some traditional methods, such as the sale of the use of individual copies and patent royalty payments, is much more difficult and sometimes impractical with open source software. Moreover, many see the introduction of open source software as damaging to the market for commercial software. Most software development companies sell licenses to use individual copies of software as their primary source of income, using a combination of copyright, patent, trademark and trade secret laws (collectively called intellectual property rights laws). Fees from sale and licensing of commercial software are the primary source of income for companies that sell software.

Additionally, some companies with large research and development teams develop extensive patent portfolios, with the purpose of making money from patent royalties. These companies can charge licensing fees for the use of their patents in software, however open source distribution creates the potential for an unlimited number of derived works using the patented technology without payment to the patent holder.

This complaint is countered by a large number of alternative funding streams, which are actually better-connected to the real costs of creating and maintaining software. After all, the cost of making a copy of a software program is essentially zero, so per-use fees are perhaps unreasonable. At one time, open-source software development was almost entirely volunteer-driven, and although this is true for many small projects, many alternative funding streams have been identified and employed for open source software:

- Give away the program and charge for installation and support (used by many Linux distributions).
- "Commoditize complements": make a product cheaper or free so that people are more likely to purchase a related product or service you do sell (this is a primary reason for OpenOffice.org; Sun gives away the office suite to encourage users to buy their computer hardware). This is similar to The Gillette Company giving away razor handles so they could make money on razor blades, or Radio Shack giving away :CueCat scanners.

- Cost avoidance / cost sharing: many developers need a product, so it makes sense to share development costs (this is the genesis of the X Window System and the Apache web server).
- Increasingly, open source software is developed by commercial organizations. In 2004, Andrew Morton noted that 37,000 of the 38,000 recent patches in the Linux kernel were not created by developers directly paid to develop the Linux kernel. Many projects, such as the X Window System and Apache, have had commercial development as a primary source of improvements since their inception. This trend has accelerated over time.

Additionally, it is worth noting that open source programmers may have non-financial reasons for developing software. An analogy is that of Wikipedia, where people contribute without expecting compensation.

4.3. End user support

Computer software is complex enough that users frequently need help with it even after they have got it set up and working to begin with. Software also invariably has bugs in it, which may adversely impact the users' ability to get work done and so need to be fixed. And a user may see areas in which the functionality of the software may be improved, to help not just that user but others as well.

Closed-source software vendors typically provide a "one-stop shop" for all support matters: since the vendor developed the software (and appropriately licensed any included components that were developed by others), the vendor also provides all necessary support functions. Nobody else can provide the level of support that the original vendor does, simply because nobody else has the requisite access to the source code (not just to understand how it works, but to make modifications and fix bugs).

This kind of model works well up to a certain point. However, as the number of customers increases, its effectiveness decreases. The better-known PC software packages of today have customer bases numbered in the millions. With that many users, support needs to be delegated to a group of people separate from the software developers (otherwise the developers would have no time to work on the software). This group increases development costs, and invariably, not all bugs may be fixed fast enough to remain profitable. Another problem when a closed source project is this big is hackers trying to compromise other users' systems, often outnumbering the bug-fixers. (However, some say that it is easier to find bugs in open source software, as bugs can be more easily found with source code.)

Open-source offers an alternative model, where easy access to the source code allows the proliferation of a multitude of alternative support organizations, each remaining small enough to remain responsive to the needs of its own set of customers. With ready access to the source code, anybody can find a bug or shortcoming in the software, and submit a patch

for it all the way back to the original software developers, who in turn can very quickly decide whether the patch is worth accepting or not. It is often said that the more eyes looking for bugs reduce them, and with more people looking for bugs, then looking to exploit them, it is no wonder bug fixes may be faster for open source projects.

4.4. Innovation

Open-source software has often been accused of being more derivative than innovative. This is true to some extent, mostly in the desktop arena. For example, GIMP is in many ways a reinvention of the functionality of Photoshop, while OpenOffice.org is primarily designed as a plug-compatible replacement for Microsoft Office.

Many of the largest well-known open source projects are either legacy code (e.g., FreeBSD or Apache) developed a long time ago independently of the open source movement, or by companies like Netscape (which open-sourced its code with the hope that they can compete better), or by companies like MySQL which use open source to lure customers for its more expensive licensed product. However, it is notable that most of these projects have seen major or even complete rewrites (in the case of the Mozilla and Apache 2 code, for example) and do not contain much of the original code.

However, one should not overlook the many innovations that have come, and continue to come, from the open-source world:

- GCC is a set of compilers for C and other languages, that supports more different processor architectures, for both native and cross-compilation, than any other compiler.
- The Linux kernel is available for nearly two dozen different major processor architectures – more than most other operating systems.
- Mozilla Firefox is a Web browser which has managed to take increasing market share from Microsoft's Internet Explorer, to the extent that version 7 of Internet Explorer offers many features similar to those already in Firefox. Firefox, however, copies many features from closed-source browsers such as Opera.
- Beowulf MPI is an open-source framework used for building parallel-processing applications that run on Linux and other UNIX-like operating systems. It has proved itself so powerful that Microsoft has adopted it as a crucial part of its own efforts to establish a presence in the supercomputing market.
- The Gmail Filesystem is a good example of the collaborative nature of much open-source development. Building on FUSE (which allows filesystems to be implemented in userspace, instead of as code that needs to be loaded into the kernel) combined with libgmail, which is a Python library for programmatic access to a user's Gmail

message store, the result is the ability to use the multiple gigabytes of Gmail message space as a fileserver accessible from anywhere on the Internet.

- Perl, the pioneering open-source scripting language, made popular many features, like regular expressions and associative arrays, that were unusual at the time. The newer Python language continues this innovation, with features like functional constructs and class-dictionary unification.
- dcrw is an open-source tool for decoding RAW-format images from a variety of digital cameras, which can produce better-quality output than the closed-source tools provided by the camera vendors themselves.
- Nicholas Negroponte's \$100 laptop will use Linux as its operating system. The decision was made after months of discussions with vendors of closed-source alternatives.
- A number of laptop models are available with a particular emphasis on multimedia capabilities. While these invariably come preinstalled with a copy of Microsoft Windows, some of them also offer an alternative "fast-boot" mode based on Linux. This gets around the long time it can take to boot up Windows.
- JACK is an audio-routing architecture that has been found sufficiently useful to be ported to Apple's Mac OS X.

4.5. Compatibility and interoperability

One issue that needs to be watched with software as it evolves is the need to maintain compatibility with other software, whether via common data formats, communication protocols or APIs. Closed-source software has a particular problem with APIs, since old, obsolete ones may need to be supported essentially forever, because they are still being used by other software that has never been updated.

For instance, consider the migration from 16-bit to 32-bit x86 processor architectures. Microsoft brought out its first 32-bit version of Windows, Windows NT 3.1, in 1993, yet it wasn't until 2001, with the release of Windows XP, that it was able to put out a mass-market 32-bit OS that did away with most of the backward compatibility with 16-bit code. Now there is the next transition, to 64-bit processor architectures. Windows XP Professional x64 Edition was released in 2005, but full 64-bit support in the way of applications and hardware drivers still remains thin on the ground, and the migration looks likely to be at least as protracted as the 16-to-32-bit one.

Contrast the situation with Linux. That has been available on a range of processor architectures, including 64-bit ones, since early in its history. Now that 64-bit processors are

commonplace in the mass market, all the major Linux distributions offer full 64-bit-native versions, running full 64-bit binaries and using full 64-bit drivers.

4.6. User freedoms to exploit software

The politics of Open Source is an area that is so controversial, often even Open Source advocates themselves cannot agree what it should be.

The free software movement demands the following four freedoms:

- Freedom 0: The freedom to run the program for any purpose.
- Freedom 1: The freedom to study and modify the program.
- Freedom 2: The freedom to copy the program so you can help your neighbor.
- Freedom 3: The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

However, one thing all are agreed on is that Open Source software is never designed to prevent users from doing things they might legitimately want to do.

For example, when Adobe Systems released its SDK for Photoshop 7.0 in 2002, it abandoned its previous policy of making SDKs available for free download from its Web site, instead requiring prospective developers to sign non-disclosure agreements before they could obtain the SDK. The reason given was that the new version included trade secrets and other commercially-sensitive information that could not be revealed to all and sundry. Which seemed reasonable enough. Except that, at the same time, SDKs for 6.0 and earlier versions of Photoshop were withdrawn from the site. Surely the argument of trade secrets in version 7.0 could not retroactively apply to earlier versions. Yet this kind of arbitrary exercise of control is precisely the sort of thing that regularly happens in the closed-source world, and that open-source advocates abhor.

4.7. Integration and overall "feel"

When people compare the user experience with using Microsoft Windows versus typical Linux distributions as a desktop system, they generally agree that Windows works in a more seamless fashion. Every bit of the system was produced by one company, so naturally the parts work together well. A typical Linux distro, on the other hand, is a combination of pieces from a large number of independent groups: the Linux kernel itself, basic operating system infrastructure from the GNU Project, basic GUI functions from X.Org on top of which one may run window managers or alternatively more elaborate GUI environments such as GNOME or KDE, and so on.

But on the other hand, all the different open-source groups have a strong interest in having their projects work well together. They achieve this by having a fondness for open

interoperability standards, such as those promoted by Freedesktop.org and the Linux Standard Base.

This cooperation between different groups naturally has to be conducted on a more structured basis than that between different departments of the same company. This turns out to have benefits in some respects. For instance, the Internet Explorer browser is so heavily tied into the Windows operating system that it needs to run effectively as "superuser", with full access to every part of the machine. This kind of situation would be unacceptable in the Linux world, where ordinary user desktop software has no business requesting full superuser privileges; if there is some task that it needs superuser privileges to perform, then it can ask the user for appropriate permissions before performing that specific task, or if that is not convenient then alternatively a suitably-privileged daemon could be created to perform that part of the task, with a defined protocol for communication between privileged and non-privileged code, to minimize the opportunity for security breaches that might compromise the integrity of the machine. Having the entire application run all the time with full superuser privileges is almost never the right solution to the problem.

For another example of why integration needs to be done on a carefully-structured basis, compare the systems for applying updates to operating system installations in Windows versus typical open-source operating systems. In versions of Windows up to XP/2003, updates (patches) are applied to the OS as a monolithic whole:

One of the new features under consideration for the next version of the Windows Installer is the ability to uninstall a patch. Currently you must uninstall the whole product or use a hacky anti-patch style mechanism.

...

Currently patches are applied by MSI in the order they are received at the client, not the order they were created by the author. This can get really nasty in some scenarios, because applying patches in the wrong order can actually result in files being down-reved.

Most Linux distributions, as well as the BSD operating systems, on the other hand, include package management systems as standard. The various components of the installation are carefully separated into individual packages, with clearly-defined dependencies between them. An attempt to upgrade a package on which another package depends will trigger a message to that effect, perhaps with an offer to automatically upgrade the latter package as well. Two packages that do not depend on each other can be independently upgraded, and if a problem is revealed with the new version of one of them, it can be independently reverted, regardless of the order in which the two were upgraded.

And since the package management systems are open-source and public, it is straightforward for third parties to set up additional package repositories (such as Packman for SuSE Linux) that integrate cleanly with the original vendor/developer provided ones

4.8. Security

Open source advocates usually believe that open source programs are more secure, mostly because flaws in the code can be seen and fixed by anyone. Different studies reach different conclusions about security through obscurity versus open source. Also note that proprietary software companies may not always release advisories for all bugs in their software. Closed source advocates, including Microsoft corporation, argue that since no one is responsible for open source, there is no way to know whether it has been fixed. Open Source advocates argue back that no one knows what bugs exist in a closed source product, since there is no one independent and credible checking in depth claims made by its vendor nor any open process addressing problems whose quality can be examined by 3rd parties.

Some people believe that closed source software is more secure than open source software. With any given piece of software, it's much easier for a black hat to find and exploit security holes in any given piece of software when he has the source code than without it. For example, many open source web programs using PHP have serious security problems and although these problems are being fixed, they are only fixed when affected end users prompt the developers about the problem.

Other people believe that open source software is more secure than closed source software. The availability of open source code leads to faster discovery of security issues, and faster resolution of these issues. They point to the exploitation of proprietary software such as Internet Explorer. But others claim that such software is exploited because it has a large market share—making it an attractive target for attackers—and claim that open-source software would also be exploited if it attracted the attention of those attackers. Open source advocates often counter by pointing to Apache, which is more popular than its main competitor, Microsoft IIS, but is also exploited less often. This argument is analogous to one in cryptography: it is believed that a secure encryption scheme has to be able to withstand attacks from people who have access to the code, and that security through obscurity is not a good thing. However, cryptography and software development are very much different things.

Flaws certainly occur in both closed-source and open-source software. However it has frequently been the case that a patch to fix one security problem in closed-source software has created another problem or failed to fix the actual problem, and other times a vendor may leave a known flaw unpatched for months or even years at a time. These sorts of issues seem to be less common in open-source software

5. Criticism

Critics of "open source" publishing cite the need for direct compensation for the work of creation. For example, the act of writing a book, building a complex piece of software, or producing a motion picture can take a substantial number of person-hours. Retaining intellectual property rights over such works greatly increases the feasibility of obtaining financial compensation which covers the labor costs. Proponents argue that without this compensation, many socially desirable and useful works would never be created in the first place. Some critics draw distinctions between areas where open source collaborations have successfully created useful products, such as general-purpose software, and areas where they see compensation as more important and collaboration as less important, such as highly specialized complex software projects, entertainment, or news.

Another criticism of the open source movement is that these projects are not really as self-organizing as their proponents claim. This argument holds that open source projects succeed only when they have a strong central manager, even if that manager is a volunteer. The article *Open Source Projects Manage Themselves? Dream On.* by Chuck Connell explains this viewpoint. Eric Raymond responded to this criticism [here](#), and Chuck Connell answered [here](#).

The Free Software Foundation (FSF) opposes the term "open source" and the professed pragmatism of the open source movement, as they fear that the free software ideals of freedom and community are threatened by compromising on the FSF's idealistic standards for software freedom

6. References

- Open Source [*Wikipedia*]
http://en.wikipedia.org/wiki/Open_source
- History of Open Source [*Wikipedia*]
http://en.wikipedia.org/wiki/Open_Source_history
- Open Source vs. Closed Source [*Wikipedia*]
http://en.wikipedia.org/wiki/Open_source_vs._closed_source
- The GNU Manifesto [*GNU*]
<http://www.gnu.org/gnu/manifesto.html>
- Open Source Definition [*LINFO*]
http://www.linfo.org/open_source.html
- Jon Ippolito, *Why Art Should Be Free*
<http://www.nothing.org/osc/WhyArtShouldBeFree.htm>