Open Source

Open Source

http://en.wikibooks.org/wiki/Open_Source
This Book Is Generated By Wb2PDF
using
RenderX XEP, XML to PDF XSL-FO Formatter

19-04-2011

Table of Contents

Open Source

This book is intended to be an introduction to the Open source software movement. It is still getting organized. At the moment a basic outline is available below, once there is some content for each section, and it looks like the outline is more or less stable, the various chapters will be moved into proper sections to conform to Wikibooks standards. Please take a moment to read the introduction to understand the direction this book is intended to take (at least as of this writing).

Cover Art Created by Lobster

Introduction

In the last few years the Open Source movement has changed the face of computing more than almost anyone would have thought possible. Many people in the computer industry believe that these changes are the most important changes in computing since IBM contracted Microsoft to write an operating system for their personal computers. This book is intended to discuss the history, philosophy, and legal issues relating to Open Source, as well as ways for software developers to get involved by selecting and joining projects they want to contribute to. More detail will be provided here than the Wikipedia articles on Open Source and Free Software, and information found in the Wikibook on Freeware will be enhanced and extended. Please feel free to add/correct information throughout the book. This book is **not** meant to be a listing of projects nor a guide to picking the best software to use (although some pointers are provided later in the book).

What is Open Source?

Open Source or *open-source software* is different from *proprietary* software. In Open Source, the source code used in the software is available to *anyone* to examine, evaluate, and adapt. Open source has had an important impact on the way many developers view and create software. End users often use the term open-source to cover a variety of *free and open source software*.

The Open Source Initiative has this for a definition of open-source software:

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

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.

Rationale: By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.

2. Source Code

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. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

Rationale: We require access to un-obfuscated source code because you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Rationale: The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source

code. The license may require derived works to carry a different name or version number from the original software.

Rationale: Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.

Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

Rationale: 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.

Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licensees of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Rationale: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

Rationale: This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

Rationale: This clause forecloses yet another class of license traps.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

Rationale: Distributors of open-source software have the right to make their own choices about their own software.

Yes, the GPL is conformant with this requirement. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

Rationale: This provision is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called "click-wrap" may conflict with important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that (a) redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that (b) the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.

Free software is open-source software, but not all open-source software is free.

What is Free Software?

Free software is software that adheres to the concept of the four freedoms of software, as articulated by the GNU Project. These four freedoms are as follows:

- The user is free to use the software for any purpose.

- The user is free to study the mechanisms by which the program operates and to adjust these mechanisms to a specific purpose.
- The user is free to redistribute the software to another user.
- The user is free to adapt and improve the program and release these adaptations and improvements to the public.

Free Software shares much of its philosophy with Open-Source software, but many people within the open source community feel that there are important distinctions between the terms, as described in the section Free Software vs. Open-source software.

Often Free Software is referred to as "free as in speech, not as in beer", stressing the idea that the Free-software movement is concerned with *freedom*, not with *price*.

Free Software vs. Open Source software

The primary distinction of open source software is that it's not about freedom, it's about what software does things better.

This requirement is not true of all open source licenses. In this book we hope to provide enough information so that the reader can understand what the two are and make choices accordingly.

As a whole the movement is often called *"Free and Open-Source software"* (FOSS). While many people point to the differences between the two, this book will focus on what unites the two. For the purposes of clarity the abbreviation *FOSS* will be used when describing issues that apply to all open source software projects, whereas *OSS* will be used when describing issues that apply only to software with open-source licenses that allow future developers to close the source code, and *FSS* will be used when talking about issues that apply only to software with free licenses requiring future developers to maintain the previous style of licensing (if not the license itself).

Public Domain Software

Another common form of open-source software is software that has entered into the public domain. This is software that is unrestricted by copyright or licenses, and therefore free to use for any purpose. Before the rise of the open-source software movement a great deal of software written in academic circles was released into the public domain for peer-review. This practice has changed with the rise of a more intentional open-source movement, resulting in much software that was once public domain, now being released as FOSS. The public domain is where many resources used in

open-source projects come from, but there are few (if any) major projects that operate totally within the public domain.

FOSS does not mean no cost

This is a common misunderstanding about FOSS, in no small part because nearly all FOSS programs are available free of charge. For example when the text editor Emacs was first released Richard Stallman charged from time to time to get copies. Developers have the choice to charge under most FOSS licenses, although they rarely choose to. The only requirement to be a truly FOSS project is that the publisher provides the source code with the program, and to allow the user to edit that code.

On top of the initial cost of purchasing software, there are other ongoing costs associated with all software. This can come in the form of support agreements, the cost of customization, training costs support personnel and other sources. This is true of both traditional commercial software and FOSS programs. There is a large and active debate about which type of software is more expensive over the long run for large corporations, for individual users there is little to no question that FOSS is cheaper by far.

History

The history given here is not intended to be a complete, and perfect rendering of the history of the open source movement. Indeed it would be quite impossible to do so, as we are currently in the middle of the movement, and proper histories cannot be written about current events. Instead this chapter is intended to give the reader a general understanding of how the open source movement got started, so they can better understand the current debates.

In the beginning (1960's and 1970's) nearly all code was provided as a form of open-source. Since you had to get programs to run on many different machines, companies that wanted customers to be able to run their software they had to provide the source code to compile themselves. Frequently this code was not encumbered by licensing agreements, or those licenses were never enforced. Often the users would notice flaws in the programs, fix them, and provide the improvements back to the publisher at no charge. Unix user groups often shared code with each other as well, creating an even larger body of code to work from. At the time most companies believed that hardware would always generate far more profit than software.

At the same time when academics wrote programs as part of their research, they would often release the code into the public domain to others to learn from their work. The first email server, ftp

server, and web server were all public domain projects that were created by academics and shared as public domain software.

Over time the hardware became more standardized. This allowed software publishers to tighten up on what segments of code they would allow their users to see and edit. This tightening started to frustrate users, especially academics, since they could no longer fix the problem they found, and from time to time found that they couldn't use the hardware or software they wanted to. This trend mostly continued in main stream software, strengthed by the rise of Microsoft.

During the late 1970's, many small businesses attempted to survive writing software for the new microcomputers. None were able to rely on hardware sales or after sale support contracts to subsidize software development. Few of these people, whether from an academic tradition of freely distributed software or not, were willing to make their products available without fee. Perhaps most famously, Bill Gates (a co-founder of Microsoft) realized that software, not hardware, held the prospect of being the greatest profit source in the new world of computing. When IBM needed an operating system for their personal computer, instead of writing their own, Microsoft licenced an early version of DOS, but retained the licensing rights for other hardware manufacturers. This choice changed the face of computing by creating a standard operating system that most personal computers would run on, it lead directly to Microsoft's dominance of personal computing, and later allowed them major access to the server side market as well. Microsoft has grown into the largest software producer in the world, competing in almost every segment of the market. There are many people that feel that the only way to compete with Microsoft is through open source projects which leverage a far larger support base than most companies can provide.

In this sea of activity, the FOSS movement primarily grew out of two places: the Berkeley Software Distribution license from the University of California and Richard Stallman's Free Software Foundation Gnu Public License.

BSD

During the 1970s, after Ken Thompson of Bell Labs spent a year teaching at the University of California at Berkeley, several research groups there began to develop software for the Unix operating system. And, as Bell Labs had no plans to port the system to the then new DEC VAX computer system, the Computer Science Research Group at Berkeley undertook the port. That port became very popular at other universities, and the CSRG began distribution of it, with their modifications and additions, under the name Berkeley Software Distribution. The researchers at Berkeley frequently found ways to improve or add to the existing versions of Unix. Bell Labs accepted some of the BSD changes, and passed on others. Over time the CSRG found ways to improve the basic aspects of Unix (eg, the file system). Concurrently, AT&T made improvements of their own, causing the two versions to move in different (although similar) directions. As development progressed on the

Berkeley version of Unix, it grew further and further away from the work that AT&T was doing on their version (the last version of which was System V). In the early 1990s there was a major legal skirmish over the Berkeley Software Distribution (BSD) between AT&T, the University of California at Berkeley, and the Unix System Laboratories (USL), to which AT&T had turned over the marketing and development of Unix. The suit resulted in most of BSD being found to be unencumbered by copyright restrictions by those who controlled the original Bell Labs code. By the mid-1990s, the CSRG released 4.4BSD-Lite, Release 2 as their final product as they were closed down by the Regents of the University of California. The BSD releases were the root of the various versions of BSD that are common today, most notably FreeBSD, NetBSD and OpenBSD. Throughout its development, BSD was distributed in an open manner, resulting in contributions from many different contributors.

The Free Software Foundation and the GNU Project

In 1983 Richard Stallman (then of MIT) became frustrated with the growing commercialization of the computer development work that had been done at MIT, and with the increasing limitations imposed on software users. After a time, he began to create software that gave control to users. His vision was to create an entire operating system totally free of the restrictions being imposed by proprietary licensing. His first major software development effort (even before he became disenchanted with trends in the software industry) was the text editor, GNU Emacs. His next was a LISP system, a w:C compiler, and finally the GNU project.

In 1985 Stallman founded the Free Software Foundation (FSF) to help generate support for the GNU Project. FSF has grown into one of the most important organizations in the FOSS movement. While the primary FSF mission continues to be the completion of the GNU operating system, FSF has also taken on the role of *"free-software evangelist"* by protecting and supporting free software. FSF also holds the copyrights to much of the source code written for the GNU project, ensuring that it always remains freely availabe to users.

Over the next 10 years Stallman gathered a group of people together who essentially developed all of the core utilities found in Unix and the Unix-like operating systems. In the Unix spirit, these consist of hundreds (perhaps thousands) of small utility programs and tools. This project is still operating and is known as the GNU Project. Many of these programs have become standard on the BSD variants as well (see above). In 1994, Linus Torvalds released the first version of the Linux kernel, and when combined with the GNU utilities already available from the GNU Project, the Linux operating system came to be. The Debian distribution of Linux is called GNU/Linux, in recognition of Stallman's position that it is a joint production, to recognize that the GNU Project provides most of the essential utilities. The GNU Project continues its slow progress on its own operating system kernel, which will be known as Hurd. Hurd is intended to be offered as the official GNU replacement for the Unix kernel, though it is currently at a beta development stage.

While the FSF has become somewhat controversial, it has certainly had a major impact on computing. Millions of copies of the GNU software are used every day throughout the world.

For More Information

Much of the information for this article came from the following sources. They also contain more information about their areas of open source history if you are interested.

Wikipedia Articles:

- Free software article's history section

Outside Sources:

- Official GNU project History
- History of BSD

Philosophy

There is no one philosophy that brings people to open-source software. There are however two main branches of thought in which many people share some similar views and have helped create a strong movement. To understand the open-source movement as a whole you need to understand both of these main philosophies.

Software Freedom

An earlier software movement similar to Open Source is the Free Software Movement (embodied in the Free Software Foundation). Although there is some debate over its precise definition, there are generally three primary components of software freedom.

- **Freedom to Use**

Many believe that software should not have limitations on their private use, aside from restrictions on the creator's liability. In contrast, some commercial packages (like Microsoft Windows) disallow certain usages while a more expensive product allows such a use. According to US copyright law, this right is implicit unless limited by private contracts.

- **Freedom to modify**

Possibly more important than the freedom to use is the freedom to modify existing programs. Although certainly related to the overall open-source ideal, this freedom also implies that your modifications remain yours. Some open-source licences require the modifier to assign the rights to their work to the originator of the license (Microsoft's Shared Source program is generally considered one example). Obviously, very few completely commercial software packages give users this freedom. In US copyright law, this is an implicit exclusive right of the creator of a work. To give others this right, the creator must agree to do so.

- **Freedom to distribute**

A final key freedom is the ability to distribute software, modified or unmodified. Software packages that limit their distribution limit others freedom to do with it what they will. It can also prevent them from providing their improvement to the community. Similar to the freedom to modify, some open-source licences also limit the freedom to distribute, often requiring changes to be submitted back to some central repository. As with the right to modify, this is an exclusive right of the creator.

Notably absent from these freedoms is the requirement that free software be economically free. Several licences do not prevent merchants from selling the software whether modified or unmodified. In fact, several groups have historically charged for various pieces of software which are generally considered free software. Depending on the licence, the merchant may still be under other restrictions.

An example of a licence that provides software freedom with some restrictions on the distributer of the software is the GNU General Public Licence, or GPL. The GPL was written by Richard Stallman et al., for the GNU Project. The GPL has all of the above freedoms listed within. The most important difference from the above baseline is what the Creative Commons calls a "Share and Share Alike" clause. If you distribute a modification to a particular software package under the GPL, then you must agree that your portion of the work is also licenced by the GPL. For more information about the GPL there is more information in the licenses section of this book

Open Source Development Model

The second main branch of open-source philosophy revolves around the opportunity for a new software development model. Open-source is also about sharing ideas, and spreading the effort of creating software over a large number of interested developers. This sharing can lead to better software in shorter development times. It also allows for better feedback directly to the developers than is often present in traditional development models. Major companies like Apple and IBM have recognized that there can be a significant advantage to writing software in this way. Apple by way of the

Darwin project and IBM through several significant donations of software and patents to open source groups have paved the way for other companies to seriously consider using this development model.

Licences and Patents

What Is A License?

In legal theory, a **license** is the permission or right granted to engage in some act without which the act might be otherwise unlawful. A license must be granted by a person, institution, or government that has the legal authority to do so. The term "license" also refers to the document that specifically describes these permissions and rights.

Important FOSS Licenses

While there are many open source licenses in use today, a few have emerged as more popular (and therefore more important) than most. We will take some time here to examine these important licenses in some detail. They are listed here in alphabetical order.

For an extensive list of open source licenses see the GNU Project's list of licenses, or The OSI list of licenses.

Apache

The Apache Software Foundation (ASF) is a non-profit corporation to support Apache software projects. Apache's projects are characterized by a collaborative, consensus based development process, an open and pragmatic software license. Among the ASF's objectives are to provide legal protection to volunteers working on Apache projects, and to protect the Apache brand name from being used by other organizations.

There are currently two forms of the Apache in wide use today: versions 1.1 and 2.0. While 2.0 is considered by ASF to be the current version and most users have happily switched, a few large groups objected to new clauses inserted into 2.0 regarding patents, and have refused to use any software with the new license (most notable the OpenBSD project).

Code under Apache license can be linked with the proprietary code that is developed in the companies. Because of this the license is accepted inside the most of the companies. Companies time to time contribute to the code under Apache license to boost the activity of these projects. For instance, IBM and Intel have contributed big parts of the alternative java implementation to the Apache Harmony project.

BSD

BSD is both a license, and a class of licenses (generally referred to as BSD-like). The modified BSD license (in wide use today) is very similar to the license originally used for the BSD version of Unix (see BSD history above). The BSD license is a simple license that merely requires that all code licensed under the BSD license be licensed under the BSD license IF redistributed in source code format. BSD (unlike some other licenses) does not require that source code be distributed at all.

BSD is actually very close, though still distinct, from a public domain licence. Like works in the public domain, BSD-like licences permit nearly free modification and distribution of a work. There are only two significant ways in which BSD-like licences differ from public domain. First, the BSD licence carries the requirement that the licence, along with it's statement of copyright, is carried along with the work, or modified copies. This does not however limit code under different licences from being combined with it. Second, the BSD-like licences have a standard disclaimer of guarantees, such as fitness for a particular use and merchantability. This latter portion should not come as any surprise, since practically all software packages carry a clause nearly identical to this one.

In summary, the major difference between BSD-like licences and the public domain is that BSD licenced works must remain attributed to the original creator(s). Some BSD licences take this further, and require similar copyright statements in advertising materials for a distributed software product. Most however are simply limited this basic requirement.

For those who are opposed to the GPL for its limitations on modification and distribution, the BSD licence is often quite acceptable. Due to the inherent lack of limitations, BSD code can be incorporated into any other project, commercial or otherwise, removing a lot of the worries of licencing issues from code. Similarly, for those who endorse the licences such as the GPL for their ability to protect software freedoms, the BSD is looked upon as a step in the right direction, but not enough to guarantee said freedoms. They often encourage the use of more GPL like licences for this reason.

BSD is usually accepted inside the companies. The Apple Mac OS is based on the originally BSD - licensed operating system but with big parts of proprietary code.

GPL

The GNU General Public License is a free software license, created by the Free Software Foundation (FSF); version 2 was released in 1991. It is usually abbreviated to GNU GPL, or, simply, GPL. FSF recently announced that they have started work on version 3 of this license, and should be releasing the new version in the spring of 2007. The motivations for these changes have not been articulated yet, but based on a lecture by Richard Stallman held at New York's Cooper Union on November 29, 2005, the changes are related to the World Intellectual Property Organization's (WIPO) work on world copyright standardization.

The GPL protects the 3 main articles of software freedom as defined in the section on Software Freedom. The GPL also contains a clause requiring publishers of GPL licensed programs, that make modifications to the software, to distribute that modification under the GPL. This is something that many company managers do not like because they want proprietary control on all code that was written in the company. Even if the software is for internal use only and is never sold, the managers still want proprietary control to be sure that the software will not be used by a concurrent. Trying to create the negative attitude, GPL oponents gave GPL the label "viral", since it apparently "infects" any code which it touches.

From the other side, it is possible to say that GPL is tuned to support and even enforce the code sharing as much as possible. Companies and individuals that write code and release it under GPL get access to the large amount of the existing code under GPL. That way they are rewarded for picking this license and probably can afford to experiment with GPL compatible business models that otherwise bring less direct profit than proprietary control on the code.

It is not true that GPL is never accepted inside the companies: many companies have used software libraries under this license and did released all code that interacts with these libraries, as the license requires. This is how GNU gcc compiler have got C++ support, for instance. It is also true that GPL seems one of most popular licenses in the world. However the most of the companies will likely first try to contact the authors of the GPL-covered software, asking to sell the "commercial" license.

LGPL

An important corner-case of the GPL requires that only GPL code can link with with GPL libraries, even if the library is contained in a separate file, and the code which uses it contains no GPL code unto itself. This is seen by some as too strict, since such an action could be considered a "use" of the library, not as a modification. In order to allow for such a distinction, the LGPL was created. The LGPL fixes precisely this problem: If someone uses an LGPL work as a library which is sym-

bolically linked with a project, the rest of the project does not have to be put under the LGPL. On the other hand, if any modifications are made to the LGPL'ed work which are distributed along with the software package as a whole, the changes must be made public, similar to the rules of the GPL. The "L" in the LGPL stands for different things for different people. Initially it was considered as a "Library" licence, since the clause specifically dealt with issues of linking. However, the GNU Foundation describes this as the "Lesser" GPL, since it gives fewer protections to the work than the GPL does. While the GNU Foundation does not frown upon the use of the LGPL, it encourages the use of the GPL when possible.

LGPL makes a clear difference between the code that has been covered by this license (usually software library) and the external code that is just linked to the LGPL covered code. The external code can stay proprietary. However modifications inside the LGPL covered library must be shared with public.

Companies usually find LGPL terms acceptable and it is common to find public libraries with this license inside the commercial software.

GPL + linking exception

GPL + linking exception license sounds like GPL but in additional chapters explicitly allow to link the code it covers together with any other code, regardless of its licensing terms. Hence while it sounds similar, GPL + exception in practice looks more like LGPL. This license has been recently used for several large projects, including java implementation by Sun Microsystems.

A brief note about copyrights and copylefts

Software licensing should not be confused with copyright protections. Open-source licensing explicitly relies on copyright law protections and no code licensed under most of those licenses are a release to the public domain. The author of a piece software can limit the use of what he created, since the code is a written work and subject to copyright law provisions. A creator can limit use of the software as desired, since copyright is a privilege of exclusive use. Copyright protections allow a copyright holder (not necessarily an author) to control use and distribution of software via licensing. That license may allow changes (as does the GPL) or prohibit them (as do most commercial EULA agreements).

The concept of copyleft also grew out of the work by GNU and others in the open source movement. While copyright is intended to most protect written works from unauthorized use, and most commercial copyright licenses do so by prohibiting changes or distribution, copyleft licenses encourage free and widespread use.

Patents

In the past few years the issue of software patents has become one of the most pressing to those in the open-source movement. In the United States and some other countries software is subject to patent protections. While only some other countries have placed similar restrictions on the production of software the patent system in the US has become problematic worldwide.

Economics of FOSS

Zero Marginal Cost

At the heart of the economics of FOSS is the zero marginal cost of goods in a digital environment. In this respect, the emergence of FOSS represents a confirmation of classical microeconomic price theory - that an equilibrium price in a perfect market approximates the marginal cost. From this perspective, FOSS can be understood as a pioneer in reaching what can be understood as an evolutionarily stable dynamic Nash equilibrium in truly free market.

Income-generation opportunities

While contributing time and effort in developing, enhancing and documenting FOSS does not provide any direct income, the development of expertise in FOSS provides a broad range of income-generating opportunities - from generating in-house savings from enhancements to FOSS to consulting opportunties in installing, training, customizing and the provision of technical support for FOSS installations. In Part I: The Networked Information Economy of *The Wealth of Networks: How Social Production Transforms Markets and Freedom* (Yale University Press, 2006) [1], Yochai Benkler provides an excellent analysis - with IBM's stategy as a key example - of ways that income and wealth are being generated through open source and open content strategies.

Opportunities for the companies

Other possible ways of getting money from FOSS include:

- Dual license model, when the same code is provided both under aggressive FOSS licence (usually GPL) and "commercial" license that does not demand sharing of the derived works. The GPL version ensures wider user base, more popularity and may serve as an "evaluation version". This model is used by many companies, most notably by Trolltech with QT.
- Paid documentation model, when the software and convinceable examples are delivered as FOSS but some really good documentation is described in a book that must be bought, or in a separate documentation package which is for sale.
- Use of trademark. If the derived works cannot use the original trademark (logo, name) then the original work may be preferred by the user who is willing to pay for that one understands as a sign of high quality. This model is only possible if the trademark owner is already widely known as an expert in software development. It is successfully used by Red Hat.
- Use of the paid auto update service. In this case the paying user receives opportunity to get and install security and other important updates automatically. Non-paying user needs much more effort to keep the system up to date. This model is used by many commercial Linux distributions.
- Using the certification system, when the owner of the leading FOSS technology provides the paid certification exams to prove the qualification of the user (usually software developer). Many owners of the popular FOSS products (Sun Microsystems, Red Hat and others) offer the paid certifications that are recognized and valued by employers.
- Providing paid support for porting the FOSS to the new processor, device or operating system.
- Developing FOSS that at the given moment runs primarily on the device that is sold by the same company. FOSS adds the value to the device that now can attract more attention, be sold for the higher price or just simply become usable. This model is frequently used when porting Linux into mobile devices and other unusual platforms, it also enables companies to develop open source drivers for the hardware they produce.
- Using FOSS components (including components under GPL) to develop highly specialized software for internal use inside the same company. As such software is never officially distributed, there is no need to get profit from selling it. GPL does not demand to publish the source of the software that has been never released to public.
- Reaching new groups of customers. There are many FOSS enthusiasts in the world. The company that makes an open source - friendly product will likely attract these customers who prefer such products over its alternatives. An example is the FIC OpenMoko project where tens of thousands of FOSS - friendly mobile phones have been sold even while still unsuitable for the normal end user. Also, some Linksys routers and most notably the NSLU2 device have a completely unexpected group of users who were just interested in hacking their internals.

Opportunities for individuals

- For the individual developer, joining the FOSS project and contributing the code may provide expertise that is difficult to obtain in alternative ways. Software engineer that knows capabilities of the available FOSS, how to build it, how to port it and how to get the most use from it may be able to save large sums for her company. In some cases developing the FOSS replacement gives the direct expertise on the highly similar proprietary product. The notable examples include GNU Classpath and Sun Microsystem java, DotGNU and Microsoft .NET and also various FOSS and proprietary implementations of J2EE. Such expertise is the most easy to get by contributing to the existing, highly successful FOSS project.
- Participating in FOSS helps to build a professional social network: a group of people that know well enough and if needed can recommend you as a programmer.
- For the maintainers of the small but comparatively popular projects, offering the paid documentation may work, especially if this documentation is easy to buy online.

Problems with traditional commercial software

First it should be noted that the commercial software industry is one of the largest and most important industries in the world. The rise of the Open Source movement does not necessarily spell the end of the commercial software industry. On the contrary, many people argue that commercial software can be strengthed by the use of open-source techniques. Commercial software is designed to provide a product worth paying for and most of it is (that's why the industry is so big). Despite its price tag, commercial software is often far from perfect.

Commercial products will be updated frequently in order to reflect the fluctuating demands of the market and customer needs. These needs can lead to software practices that rearrange and rewrite the software too frequently, or release beta versions as commercial endeavours resulting in high rates of bugs in early versions. Some commercial programs are over designed and written in sloppy code leading to bloated, slow, running programs. Open Source by contrast is driven by the needs of the end users. The skills of the coders is in taking personal pride in what they do, not rushing to meet artificially imposed deadlines (except their own). Thereby their code is often of a superior calibre than that of programmers in the commercial environment. There is also usually a very extensive degree of feedback as the programmers test their products within a wide network of people.

When source code is available, it can be checked against various "back doors" and other security holes that may be intentionally or unintentionally left in the closed source software. In the past such holes have been found in multiple proprietary products, including software, used by the government.

Advanced developers value source code of the used components as an important additional documentation.

Having source code also means that the software can be easily ported to run under different processor, device or operating system. Proprietary software can usually only be legally ported by the original developer that may or may not treat such request as important enough to work on it.

Another problem with traditional commercial software is its closed nature. There is usually no or limited freedom to tamper with the copyrighted product. Also, companies often force users to follow upgrade paths that they may not wish to follow. Open source software enables the individual to customize the software to his end needs in all freedom. Another heavily criticised aspect of commercial software is that customers are frequently locked-in to a product because to keep using data files you are often forced to continue to use the same program. If you wish to share files with users that have upgraded, you often have to upgrade yourself or be written off as irrelevant. Since open source software allows competing programs to share different data file types, there is no reason to be trapped into any one program. If a newer version has a new file format, then there often are converters that allow users of older versions to keep up their data files up to date.

Free doesn't mean no Cost

While FOSS is free to the end user, there is a cost associated with developing the software. These costs may be smaller than developing proprietary software, because developing the project under FOSS license means that:

- Numerous web portals like SourceForge would offer web hosting, content repository, mailing lists and other essential features for free.
- The cost of advertising a FOSS project (like presenting it in the related conferences) is usually lower.
- Developing something under GPL may give free access to high quality components (like QT) that are otherwise expensive to buy or not available at all.

Still, development of any software first requires the developer time. Only very popular projects may expect to get a high quality code contributions for free.

Internationalization

The open nature of Open Source Software allows for extensive internationalization of software. OS's can be modified for any language, for example recently a version of Linux was translated into Welsh (spoken in Wales). While many major commercial projects are provided in multiple languages, it is often too expensive to translate them into more than 3 or 4 languages. Many FOSS projects are available in 10 or more languages, as the translation can be done by a programmer who wants the package to be available in his or her native tongue.

Case Studies

Probably the four best known Open Source projects in the world today are: the Apache web server, the BSD operating system, the Linux kernel, and the Mozilla web browser (or its spin off, Firefox).

Apache

The Apache web server is currently the most popular web server in the world. Nearly 70% of all web sites are served by Apache.

The Apache project grew out of an attempt to improve upon the original web server httpd. httpd, was a public domain software project that was largely abandoned in the early 1990's. At that time it was the most popular web server in use, and several developers recognized that development needed to continue. Since httpd was in the public domain they were able to continue work on the program and include a great number of improvements. They quickly organized themselves into the Apache Foundation, to have a stable organization to oversee the on-going work on their new web server. Apache's version of httpd quickly overtook the original httpd as the most popular web server in use.

Over time the Apache Foundation grew to house many related open source projects. Including Tom Cat, a popular Java serverlet server, and many others.

After many years of developing Apache 1.x, the Apache Foundation found that some of the original architecture of Apache was starting to show it's age, and further extensions were becoming impossible. Therefore a team of mostly volunteer developers started to develop Apache 2. Apache 2 is a complete rewrite of Apache, and has allowed the Apache Foundation to add new flexibility of their flagship product.

The stature of the Apache Foundation has also allowed them to become a repository for significant donations of software to the open source community. Most notably 2 large donations from IBM to the Apache Foundation in 2004.

BSD

BSD (short for *Berkley Software Distribution*) is a family of three free Unixes, namely OpenBSD, FreeBSD and NetBSD. They are all successors of BSD 4.4.

Linux

The Linux kernel is currently one of the most active, and most important, free and open source software development projects. In 1991, Linus Torvalds used the Internet to invite collaboration on his infant project to develop an operating system kernel suporting a system very much like Unix. He wanted a Unix-like operating system that would work on his computer, an early and expensive 386 machine, and was unsatisfied with anything available at an affordable cost, including academic teaching examples like Minix (from Professor Andrew Tannenbaum). Since then the Linux kernel has been used in multiple Linux system distributions that are currently supported by organizations like IBM, Novell, HP, many universities (eg, ETH-Zurich), Bell Labs, and Borland. The Linux kernel is the kernel used in the GNU/Linux operating system, and is currently in its 2nd major version release, and 6th minor version. The official home page of the Linux kernel project is kernel.org

GNU/Linux operating system is a textbook example why freedom to share the modified works is important. In his discussions Tannenbaum writes that he have frequently received suggestions to extend Minix in one or another way but was very strict when accepting these extensions. He tried to keep the system simple, easy to study and understand. People who wanted to convert it into the true next generation OS were not able to do this because direct sharing of the modified code was not allowed by the license, requiring to publish patches against the Tannebaum's version instead.

The second textbook example covers the FSF itself who have decided to refuse the available monolithic kernel and turned into difficult way of developing the microkernel based Hurd that even in 2009 have not yet been production ready. This error likely was serious enough to kill all GNU project, but FSF was protected by its own philosophy: GPL have allowed another, Linux, kernel to come, so the work spent building numerous other parts of the alternative operating system was not lost.

Mozilla (Firefox)

Mozilla and the more recent Firefox are open source derivatives of the Netscape Navigator web browser. As a result of the release of Firefox 1.0 the web saw the first major shift in web browser usage in 5 years.

GNU Classpath

GNU Classpath is a project aiming to create a free software implementation of the standard class library for the Java programming language. GNU Classpath development started in 1998 with five developers. The initial progress was rather slow because Sun was already offering its own java implementation free of charge (including commercial use) and with the source code available. Still, porting to new platforms, connecting to new virtual machines and various unusual projects required to distribute the modified versions - this was not allowed by Sun's license at that time. During the history, the project merged several times with other projects having similar goals (Kaffe, libgcj). In the past, GNU Classpath supplied its own virtual machine (Japhar). As Classpath was becoming a base library, shared with a lot of different projects, this virtual machine received less and less attention until it was dropped as no longer supported. The project started to accelerate after approaching completeness of java 1.2 API, when it was clearly seen that some companies are joining, allowing they developers to contribute to the project at working time. GNU Classpath has become a part of near any Linux distribution and was ported to numerous processors and operating systems; some of them do not have any other java support till these days. Wave of talks passed through the conferences, demonstrating for the surprised public that there are *fully functional* alternative Free implementations of Swing, CORBA and other complex libraries that - it was believed - can never be completed by the community of volunteers. The project status have changed after Sun released it own java implementation under near identical license (GPL+Exception). The primary purpose - to have the efficient java implementation under Free Software license - has been reached in another way. As both libraries implemented the same API and now had nearly identical licenses, it seemed doubtful that they can find they own groups of users rather than just taking one under another. While raise of Sun's OpenJDK meant the end of the increased status and prospects to join the history as authors of the long lived alternative java system, GNU Classpath community have shown self control keeping to serve the Free Software as good as they could: there were no opposition in the mailing lists, no attempts to create the negative image of Sun's project - shortly, no any trying to prevent replacement of GNU Classpath by OpenJDK that - now being open source - still was superior in performance. Sun has released its own libraries quite at time when GNU Classpath was about to become a fully working alternative and real competitor and it quite may be that this have contributed to the Sun's decision. Hence GNU Classpath may stay in a history as a successfull project. This project seems still active and attracts contributions, maybe because its virtual machine interface is historically much more

adapted to connect various virtual machines. It is a common choice as extension of the Google Android to provide parts that were removed from the Google implementation as "unnecessary".

Joining the existing Open Source Project

While creating your own Open Source Project is the first thing that comes in mind, the general advice for the beginner would be **do not do this**. The project must be based on a really good idea and needs cooperation of more than one developer to be really successful. Otherwise you may just observe the download counter slowly ticking up, but never get any other feedback. It is usually much better to start from joining the existing project and starting your own later, when you already have enough experience. For the same reason it is better not to start from the attempt to revive the abandoned project which has already lost its previous team (see why).

Writing and using Free software is not just a programming, is a kind of philosophy. While programming language is all you need to program, it seems also important to join the community, get friends, do a great work together and become a respected specialist with profile you cannot get anywhere else.

Steps

Initial preparation

Be sure you are competent enough to make valuable contribution as a programmer. Without this, you are useless to the developer community so nobody will talk to you. The next most important tool to master is the version control (CVS, SVN). Understand how to create and apply patches (text difference files). The most of the FOSS development in the community is done creating, discussing and applying various patches.

Searching for the first project to join

While a highly popular and very successful project may give you lots more expertise, such projects are also quite demanding to their contributors. You are expected to follow coding style, write CHANGELOG entries, create and apply patches, cover your code with tests and so on. Without experience that is expected and why, all this may end up by conflict between you and the project community. If you are totally new with FOSS, it may be more efficient to start from something less

demanding, at least for a short time. Most of the small projects that are very easy to join now can be found on SourceForge.net. The suitable project must:

1. Use the programming language you already know.
2. Be active, with recent releases. Usually a project that has no release for more than a year is considered dormant if not dead.
3. Already have three to five developers or about.
4. Use version control.
5. Have some part you think you can immediately start implementing without modifying the existing code too much.
6. Apart from the code, a good project also has the active discussion lists, bug reports, receives and implements requests for enhancement and shows other similar activities.

Some FOSS portals offerer "job openings" where administrators of the active projects are actively posting invitations for new developers. These may be interesting to read but it may be better just to search for a project you like, regardless if it posts invitations or not.

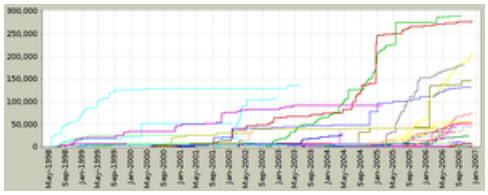
Joining the first project

1. Contact the administrator of the selected project. In a small project with few developers your help will usually be immediately accepted.
2. Carefully read the rules of the project and more or less follow them. The rules of the coding style or necessity to document your changes in a separate text file may first appear ridiculous to you. However the purpose of these rules is to make the shared work possible - and the most of projects do have them.
3. Work in this project for several months. Listen carefully that the administrator and other project members say. Apart programming, you have a lot of things to learn. But if you really do not like something, just go away to another project.
4. Do not stick with the underground project for too long. As soon as you find yourself successfully working in that team, it is time to look for the serious one.

Joining the second project

1. Find a serious, high level Free software or Open source project.

2. As we are doing a serious jump now, be ready for the far cooler acceptance. You will likely be asked to work for some time without direct write access to the code repository. The previous underground project should, however, teach you a lot - so after several months of the productive contribution you may try do demand rights you think you could have.
3. Take and do a serious task. Go on even after you discover that the task is lots more difficult than you initially thought - in this step it is important not to give up.
4. If you can, apply with your serious task to the Google's "Summer of code" to get some money from this adventure. But just do not care if the application would not be accepted as they have far less funded positions than really good hackers working around.
5. Look for a suitable conference happening nearby ("Linux days" or something similar) and try to present your project there (*all project*, not just the part you are programming). After you tell you are representing a serious Free / Open source project, the organizers frequently release you from the conference fee (if they do not, the conference is likely unsuitable anyway). Bring your Linux laptop (if you have one) and run demos. Ask the project administrator for the material you may use when preparing your talk or poster.
6. Complete the task, cover with automatic tests and contribute to the project. At this point the initial goal of becoming the active member of the serious project is achieved. You may continue in the same project or later migrate into another one to learn something new.
7. For better understanding, look into real example of the development history for a Free Software project (above). Each raising curve represents a contribution (lines of code) from single developer. Developers tend to become less active over years but the project frequently even accelerates as new people join. Hence if you already come with some useful skills, there are no reasons why the team would not invite you.



Development history

Tips

- If you still do not trust yourself enough, start from some part of code that you think is missing and can be written from scratch. Changes in existing code are much more likely to attract criticism.
- For the beginning, select a class, module or some other unit under which nobody is very actively working at the moment. Working together on the same class or even function needs more skills and a lot of care from all sides.

- Before asking any questions about the working rules inside the project, try to search for the answer in the project documentation and mailing list archives.
- Avoid asking any questions related to fundamentals of programming or programming tools. The time of the Free software programmer is as valuable as gold and nobody can afford to teach you the basic.
- For the same reason, *never* expect an older hacker writing a detailed description of your task or even providing any kind of supervision for you. While open source projects may have a lot of strict rules, they usually work along the lines of that is known as extreme programming in the programming methodology.
- Linux is a rather good system now, but hackers widely used it even when it was far less complete in the past. For a hacker it is a lot more important that all corners of this system come with source code. They are open to explore, to understand, to change and to share your changes with others. Understanding starts from the trivial daily use (writings, mail, web and surely programming). Hence you must install Linux at least on a dual boot and start from using it for routine daily tasks like web surfing or programming.
- The employers of many hackers seem motivated enough to allow contributions during their working time (usually because the institution uses the Free/Open source program that the hacker is developing). Think, maybe you can get at least part of the needed time this way.
- Always continue the hacking you started. Does not build, does not run, crashes? There *are* reasons for everything and if you have source code this usually means that you *can* force the system to do whatever you want, especially with the help of the web search. This rule has its limits, but, indeed, never yield easily.
- Only say you are a hacker after some true hacker community recognizes you as such.

Warnings

While FOSS communities are in general very friendly and cooperative, it is important not to make some trivial mistakes that may result in an unpleasant experience.

- Avoid Windows, especially if you plan to meet Free software developers. Mac OS is tolerated somewhat better, but also not welcome. If you do bring your laptop, it must run Linux or another operating system that is considered as "Free software". This may be less important in the Open Source communities but you likely will never miss by bringing Linux or a dual-boot laptop.
- If your mail client supports html messages, turn this feature off. Never attach documents that only proprietary software (like MS Word) can open properly. Hackers understand this as insulting.

- While the word "hacker" sounds respectable in most academic environments, for some uninformed people it may be associated with breaking into security systems and other computer-related crimes that a different social group, crackers do. Unless you are ready to explain, look to whom are you telling this word. Real hackers as they are meant in this article *never* join programming activities that seem for them illegal. First, they are proud of following the hacker ethic. Second, the law violations are not necessarily better paid.
- Do not volunteer to the company-owned projects that are not releasing some parts of they code under approved Open Source license. In such cases the really important parts of the project are likely to stay behind the closed doors of the owner, preventing you from learning anything useful.
- Do not start from small code optimizations, extra comments, coding style improvements and other similar "small-scale" stuff. It may attract far more criticism than any serious contribution.

Sources and Citations

- http://sourceforge.net - place to find the first project to join.
- http://www.fsf.org/campaigns/priority.html - list of the top level, high priority projects at GNU.
- http://www.apache.org/ - Apache community (they call they software "open source" and have slightly different license)
- http://catb.org/~esr/faqs/smart-questions.html - A true and useful guide how to ask questions to other hackers.
- http://freesoftware.mit.edu/papers/lakhaniwolf.pdf - A serious scientific study about Free software hackers.
- http://www.gnu.org/software/classpath/docs/hacking.html#SEC8 - Rules of the typical Free Software project

Starting and Maintaining an Open Source Project

Getting Started

Vapourware is software that is 'floated' to see if there is interest. Many projects, if donated to the open source methodology would find usage, development and markets much quicker. Lost code might be resurrected and enhanced. More and more computer users are learning to use the excellent development tools created and enhanced by programmers. If you have a good idea, open source is a way of making it happen.

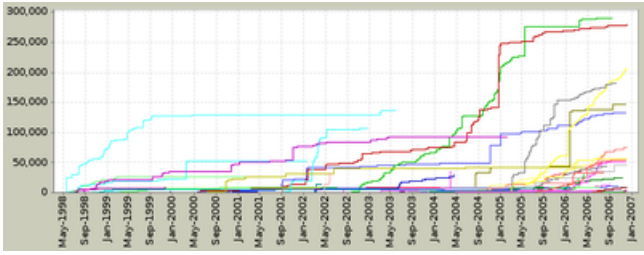
Project Management

Necessary processes of the FOSS project

There is a typical set of tools that are usually involved in managing near any FOSS project. Small projects may not use some of them; large projects near always use them all. These tools, roughly by the the frequency of usage, are:

- Version control repository that contains all past revisions of the project and can be accessed remotely. The past projects mostly used CVS. The recent projects usually use SVN or other tools.
- Bug/issue tracker that contains bug reports and requests for enhancement, together with subsequent comments and implementation status. One of the most frequently used tools there is Bugzilla.
- Mailing lists to discuss various aspects of the project development.
- Suite of automatic tests.
- CHANGELOG where all code changes are manually documented. This is especially important when there are many developers.
- Website that describes the project and provides both "getting started" and advanced documentation.

Contributors of the typical FOSS project



The real example of the development history for a Free Software project (above). Each raising curve represents a contribution (lines of code) from single developer.

If the project depends on external contribution, it is important to attract new contributors over time. Volunteers tend to become less and less active over time, and donations from the companies are also frequently not continuous. This does not mean that the project stagnates: it may become even more active over time as new developers join. However it is obvious that novices must be welcome and maintaining the small group of "elite developers" that does not allow anybody else near to the code is problematic. The software quality is ensured using other methods (automatic tests and at least brief checking of all patches by older developers).

Version Numbers

Many new users find the FOSS projects often use strange version numbering systems. For instance it is very common to see software in wide distribution that has a version number less than 1.0. For example the web browser Firefox was widely used by version 0.6, more than a year before version 1.0 was finished. In the world of FOSS software it is generally considered important to work out all the known bugs before releasing version 1.0. While this is not a totally attainable goal, many users find early versions of software much more stable than early versions of their commercial counterparts.

Generally FOSS projects break their version numbers into 3 pieces: major version, minor update, point release. Major version is the first number in the version number; it is rare to see a free software project have a major version greater then 2 (although GNU Emacs is currently on version 21, so it is not unheard of). The minor update is the second number (the x in 2.x). A change in the number often signifies a significant update of the program. The third number (the y in 2.x.y) generally indicates a small update to fix a bug, or security problem. These numbers get very high, often over 30 or 40 over time. When a software project is well established most of their updates will come in the form of point releases.

Version numbers may also carry extra meanings. For instance ever since the Linux kernel went to version 2, it has been the practice of the design team to use odd numbers for development kernels, and even numbers for stable kernels. Therefore while it is common to see 2.2, 2.4, and 2.6 all in use today, 2.1, 2.3, 2.5 are almost no where to be seen. There are still teams that work on supporting

bug/security fixes to the 2.0, 2.2, 2.4 and 2.6 kernels. As of March 2005, this policy was under review by the primary authors of the Linux kernel, however there are many projects that use similar numbering schemes, and so it holds as a useful example.

When a software project changes the major version number, this generally indicates a complete rewrite of the software, or at least an extensive over haul. The Apache web server started a version 2.0 in 2003, after many years of 1.3.x versions. Apache2 is a near total rewrite of Apache, which provides many new features, and significantly improved the underlying code base. The apache foundation still maintains the 1.3.x code base, although most efforts for new code are done on the 2.0.x code base.

Writers of software may completely rewrite a program, improve software or unnecessarily inflate perfectly good software. If software works then there is little need to upgrade.

Also upgrading may solve some problems and add new ones. To give you an example 'Abi Word' is a fine word processor but it crashed on me. The complete package in Open Office has never done this nor has notepad or WordPad. New improved 'notepads' have occasionally crashed. So I stopped being seduced by what has many features. Any software that has the hidden feature of crashing is a liability. I have used Miranda IM without problem, others find it does not work.

Open Source Hardware

Various computer hardware frequently has its own specific features that only specialized piece of the software can read properly. As a rule, the operating system contains device drivers that deal with all specifics, providing more general access for other programs. A manufacturer may choose to disclose the documentation that is required to write a driver or to keep it secret, offering the finished drivers instead.

The proprietary driver is usually a problem for the open source project. First, it usually have licensing limitations that limit its usage (being closed source is automatically incompatible with GPL). Second, there is no way to check such driver against bugs and security flaws. Finally, the driver is frequently simply not available for the platform of interest; it is quite usual to provide such drivers for only one operating system.

Because of these reasons Linux, Solaris and other open source operating systems may not provide the needed drivers for the hardware that has no sufficient disclosed documentation. In cases when such hardware is very widespread and popular, an experienced hackers may succeed to reverse-engineer the device and write they own driver without documentation. However this is a difficult and long process and the finished driver may still miss some functionality.

Hence before writing the open source software that uses some specific hardware like camera, advanced video or wireless it is always important to choose the equipment from the manufacturer that shares the necessary technical information or at least provides its own driver for your system, your platform and under conditions that are for you acceptable. This must be carefully checked in advance, without expecting that "it will work anyway" and especially without assuming that the manufacturer may change its position later.

For instance, when designing the fully open mobile phone (OpenMoko project), the FIC company have selected Hammerhead PMB 2520 chip for its first prototype, Neo1973. The documentation on accessing this chip directly has been secret. After long negotiations the manufacturer provided some driver but due licensing problems every user needed to download it from the special web site and install individually. As the phone was primarily targeted to the open source enthusiasts that do not like such "binary blobs", in many cases the driver at the end was refused, leaving the GPS on the phone non functional. Further negotiations were not successful and FIC was forced to switch into completely different chip in the next iteration. Switching, however, required to redesign some circuitry. When the new phone has been delivered, it was discovered that due lack of time the new GPS device has not been properly tested and only worked when the SD card has been removed from its slot. While at the end all problems have been fixed and resolved, long delays and non-working devices likely reduced popularity of the project that had financial problems afterwards.

Open Source Hardware is developing free BIOS and CPU specs and electronic designs.

Future Developments

FOSS is creating a new outlook. The attitude is of particular benefit to the commercial sector who obtain code and tools to commercialise. Developing nations and people can use no cost materials and resources. The developer community is trying out new systems and most importantly leaving a legacy that will be available for generations to come.

Ideas in use outside of technology

Grass roots organizers have started to take on some of the ideas generated by the Free software movement. By encouraging more transparency and sharing of ideas grass roots groups have found they can motivate more people with less effort.

Projects like Wikipedia and Wikibooks were founded with many of these same principals in mind.

Tips for picking software

There is no one best way to find FOSS projects that will best serve your needs. There are several rules of thumb you can use to help you make informed decisions.

Is this Open Source?

Some companies try to attract more users, pretending that they offer FOSS software when they actually do not. The simple way to check this is to look if they license is OSI approved. Unusual license frequently means that the project just pretends being FOSS.

Do you already have something that will do the job well?

Before you hunt up new software, make sure you do not already have a program that will do the job nicely. From the other side, having possibility to do the job does not mean that there is no software to do this more efficiently so it may be good to evaluate alternatives time to time.

Does the program do what you want?

While this might seem obvious, every IT support person can tell you stories of users that came to them asking for help using a particular tool because a friend recommended it, but it is the wrong tool for the job.

Do you care about the ideologies behind free software and open source software, or need to work with people who care?

There are many people that feel that using free software is an important part of how they live and work. Such people will not replace FOSS program by proprietary regardless of the quality of the two. However recently many FOSS is not much worse than proprietary alternatives and some is better.

Do you know other people that already use it?

Online reviews are great, but there is still nothing like talking to someone you know about a program they use every day. Real users can tell you how the program has performed over the long run. Your friend may have discovered that while it worked great at first it was missing an important feature for a project you have in mind. Also if you are doing a common task with a program that no one else is using, there's probably a reason (and it's not likely that you're smarter than everyone else...sorry).

Is the current version over a year old?

If yes, then the project has most likely died or stagnated. Active projects tend to produce new updates every few weeks or months. While old releases may be a sign of a well written program,

that needs little to no updating, often it is a sign that the programmers have moved on to other endeavours and abandoned the project.

Are there user support mailing lists you can join?

Particularly when you are new to a program it is nice to be able to get support learning your way around. Email lists or web discussion boards are often excellent sources of support for FOSS programs. Also look at the list's archives, if they are active lists you can get a sense of what the common problems, and how many people are using the program.

Other Frequently Asked Questions

Is Open Source software always of better quality than other types of software?

This depends strongly on that do you understand as the main requirements. But, in general, far from it. The same is true of any kind of software (semi-free, proprietary, etc.). Everywhere it is possible to find bad software and good software.

Programmers may still be learning, and provide their projects in an open-source venue as a means of quicker development by opening the project to others. Lack of interest may result in abandoned projects. If the need and idea is good, open source is rapidly and effectively developed. The original developer may benefit from their experience as well as faster bug discovery by others in the community. Expect many minor releases from Free and Open Source software.

Is Open Source always of worse quality, then ?

Some managers think that due lack of funding FOSS just cannot be as good as the proprietary alternative and is only used because it is cheaper to get. In recent days, it is also not necessarily so, especially taking into consideration that the criteria may be different. It is possible to propose acceptance criteria where Linux beats Windows (there *are* tests that run faster and at least less viruses), Thunderbird beats Lotus Notes (less capable but way faster and also cross platform) and KPdf beats Acrobat Reader (at least starts faster). Software like Linux kernel, OpenSolaris operating system, Sun java implementation, Firefox, Thunderbird, Google Android and many others have been created investing huge amount of money in the past. It is a myth that open source is developed only by scarce isolated individuals so there is no any particular reason why it should always be worse.

Where can I find Open Source Software?

The most reliable way to get such software is to download it from the popular, recognized web portal. The situation changes over time, but the most popular portal currently seems being source-forge.net. Also see www.gnu.org for a list of projects sponsored by the Free Software Foundation. Linux users usually get lots of the Open Source software with their distribution and can search for more with user friendly software management tools that are part of the operating system.

There are also Freeware sites that offer trial, restricted or even fully functional versions of some software for free. However it is important to understand that offering software for free does not make it Open Source by itself.

Other Resources

- Article on why FOSS works

Contributors

- Lobster: started and this book
- ahc: Took over/reshaped when the book was mostly abandoned. Started work fall 2004
- User:audriusa: Extended a little bit in 2008.