

State of the Software Industry and the Emergence of Open Source

Elmo M. Recio
uerecio@mcs.drexel.edu

Abstract

This document will outline the underlying reasons why all end-user software packages ought to be Open Source. It will explain the differences between free software and open source as they relate to the open software movement now occurring within the computing industry. It will note the reasons why more and more companies are unable to grasp this concept under the current traditional *economic principle*. This document will also explain that this concept will not infringe on the current ability to capitalise on the software developed using the open software system, supporting the idea of open software program development.

Purpose

All end user software products should be Open Source. The recent turns of events for the need of a more stable system and an increase in the necessity for cross platform compatibility in software products have reawakened the concept of Open Source. This concept is not a new, and novel idea. Before personal computers controlled the market, computer scientists would freely exchange algorithms and source code for a critical evaluation on possible improvements. Improvements would be suggested on how to make a particular algorithm stable and effective. If the evaluator was familiar with the project he would offer *patches* that would be applied to the source code by the original software developer (the maintainer.) The beneficiary of this process was not only the maintainer, but the users of the software package. This process yielded increased security, and stability for the software package (by the scrutiny of pieces of code which would otherwise go unnoticed.) Given enough eyes, all bugs are shallow.

Free versus open

A distinction must be made first off to denote the language used when referring to open software. I explicitly selected to use the term *open software* rather than *free software* due to confusions that may arise in the dictionary definition of the word free. When the term *free*, as used in the name 'Free Software Foundation' (FSF,) was originally adopted, it meant that the software source was comprised of and developed with code which was open to public scrutiny. This code was considered *public domain*. However, as more and more companies release software under the term of being 'free' they mean something entirely different.

Free as used by most modern software monoliths (Oracle, Corel, IBM, Microsoft to name a few) means that the pre-compiled software package is distributed without a charge for a copy of the software package, usually under

extremely restrictive licensing conditions as well. With a software package being distributed as machine language (binary) code, it disallows the ability for the general public to look at the source. To avoid confusion, the reference to such programs was left as free. However, the other programs that would be developed, such that its source code is public domain, were renamed to open.

Open software is a seemingly increased descriptive term for the software package developed under the concept of *public domain*. Furthermore, an Open Source software package does not necessarily denote a free copy. A company may create, test, and announce a request for comments for a piece or version of software that is currently in development. When the software is fully developed, the company may choose to release the package to certain individuals, whom have paid a fee for its use.¹ In the end, however, the final objective is met. The software development company was able to release a software product that myriad of developers (affiliated and unaffiliated with the company) was able to look at from the inside out. A project where external programmers were able to submit comments or code patches to improve bugs that would normally have not been picked up by the developers creating the product.

The Virtue of Impatience

The one major advantage of the open source paradigm is the ability to have members of the computer industry unaffiliated with the project see the source before it's *officially* released. This is done so that any existent bugs can be resolved simpler, and quicker, before the software package goes to press. To people who have been using Linux the virtues of early, and buggy releases are apparent. Attempting to convince someone from the world of Windows or Macintosh of this virtue is somewhat a different matter altogether.

A software project may be developed with six months hiatuses between releases. Good examples of this scheme are the Netscape/Mozilla Project, and the Windows 9x projects. If a user has a particular problem with the current release of the software package he must wait six months until the next time that the software is released².

Jamie Zawinski, ex-employee number twenty, at Netscape Communications Corporation, states that one of the major problems with the Mozilla Project was that they had not produced a usable web browser even after one year of development. They had not 'released often', nor released early.

This may be seen with contrast to the Linux operating system (OS) kernel development. A new kernel release is distributed every four weeks on average. Each minor release provides bug fixes that were demonstrated in the previous version. In most cases not *all* bug fixes are implemented. However, there are just

¹Some strong advocates of the Open Software Movement believe that even this is something which should not be allowed. However, if Open Source is to take the market by full force, this alternative may be the only viable source for a company's transition from closed proprietary source to Open Source. See "The Agorics Papers" re: pay-per-use versus pay-per-copy sections.

²Zawinski, Jamie. "Resignation and Postmortem."
<<http://www.jwz.org/gruntle/nomo.html>> May 23, 1999

enough differences between the previous minor version and the current version to warrant the release. Even so, the release, needing approval by Linus Torvalds himself, is not released fast enough to a community hungering for the bleeding edge. So the lead developer of the Linux kernel, Alan Cox, posts a *pre-release* version usually suffixed by his initials (AC).

With each release, the kernel developers get live and current feedback from the users and implement these changes to each subsequent version. In this manner Linus is treating his users as 'co-developers' in the project. In effect: "Release early, release often, and listen to your customers."³ This is the traditional problem with other methods of development.

Listening to your customers has a "positive" spiral effect on the development of an Open Source project. This is demonstrated by treating each of your *users* as an integral part in the Open Source development paradigm. Listen to what they have to say and let the users determine the direction of the software package.

By implementing the solutions as rapidly as possible, utilizing the internet as the main distribution medium, and releasing *early* versions of the project the users are kept satisfied. This kind of environment keeps an upward spiral of positive feedback that ameliorates the growth in the quality of the software product. This, in effect, keeps your users active within the software project.

This is one concept that the developers of Netscape (code named Mozilla) and Windows did not take into account. There is no instant gratification for users, and possibly contributors, of the software package. If there is a bug in the software package the users must wait until the next scheduled release of the software update.

Having open source software, one is able to get instant feedback and comments regarding the software package. In some cases, this feedback is accompanied with a patch for the source code that the developer may wish to incorporate as part of his software package, or implement a tidier set of code which has the same end result.

Robust, Scalable, and Ubiquitous

Open source software development offers a level of software quality that may warrant the adjectives robust, scalable, and ubiquitous. A prime example is the Internet. Currently, millions of people use the Internet daily, whether they know it or not. The grass roots of the Internet emerged as a research and development project for the military. Eventually it moved over to a network of academic researchers funded by the National Science Foundation, called ARPANET.⁴

The developers of ARPANET created standardized protocols allowing for solid and unrestricted communication possibilities. For each protocol they

³Raymond, Eric. "The Cathedral and the Bazaar."
<<http://www.tuxedo.org/~esr/writings/cathedral-paper.html>> May 24, 1999.

⁴ Berger, Robert. "Microscared: The Challenge of Open Source Software." Linux Magazine Spring 1999:27.

published a request for comment (RFC) document. These protocols were wide open for the general public and not limited in scope to a particular operating system or computer platform. By publishing the protocol and separating the design from the implementation they allowed for the free sharing of ideas and proliferation of its use.

This may be contrasted with the AT&T Bell Labs closed doors design of the Open Systems Interconnection (OSI) standard in the 1980's. It was a vague and cryptic design at the time of its presentation for use on ARPANET. It is, today, still being designed.⁵ And is probably one decade away from being completely finished. Some of the major criticisms against the OSI protocols are, in fact, due to its being closed source.

At the root of all of the following criticisms⁶ it seems quite apparent that had it been created as public domain, many of these concerns may have been addressed before standardization of the protocol.

- ◆ OSI protocols haven't been tested widely before *having been standardized*. They are not based on existing practice in large scale computer networking, like the Internet
- ◆ OSI standards are (compared to Internet standards and RFC's) *very expensive and difficult to obtain*.
- ◆ The OSI reference model is *too complex* and has too many layers.
- ◆ Having to have to understand the very difficult documentation isn't very motivating.
- ◆ Promising new network technologies like ATM networks don't fit in the OSI reference model very well and many important techniques like LAN's, RPC and stateless protocols became popular after the OSI reference model had been standardized.
- ◆ Having two completely *incompatible alternative protocols* (CLNP and X.25) *at the network layer* (and consequently many different transport layers that try to compensate for the differences) isn't what helps you in building up a fully interconnected easy to use and maintain global network.
- ◆ There is broad agreement among *knowledgeable people* that a connectionless network layer is technically superior to the X.25/TP0 approach.

The Agoric Catalyst

In order for software developers to move from the traditional "Cathedral"⁷ method of software development to an open source method, they must take advantage of the tools at their disposal. This means the need of a public forum

⁵"OSI Protocols." <http://ken.slctech.org/miscdatcomm/osi_protocol.htm> May 23, 1999.

⁶Ibid (Section 7)

⁷In reference to the closed source corporate development model as opposed to the 'bazaar' open source Linux development model in Raymond's "The Cathedral and the Bazaar." Section 1.

where information may be exchanged freely in a many to many relationship. These tools are alive and being used for this purpose this very moment. Some of these tools are the world wide web, FTP, CVS, and at the root of it all The Internet.

Linus Torvalds is not a genius, although it *does* take a lot of know how to develop an operating system from scratch, and not many of us have done that. What he *did* do, however, was take advantage of help from wherever he could muster it. This meant utilizing the Internet as a tool for developers, from all parts of the world, to be able to communicate, share ideas, and work on such a large project.

Torvalds had a simple approach to engineering his project. Open the source for all eyes to see and listen to what the users had to say. Implement as many changes as possible without regard to the additions' stability. The end goal was based on one assumption: *if you have enough users and developers, most problems will be detected early and weeded out.*⁸ This is something that can only be accomplished with the Open Source development model.

The number of developers in the Microsoft Corporation for the Windows operating system is assuredly in the hundreds. However, they are all of one mindset, and physically cultural location; Their schema, is also quite similar, and if not at first, their tied interpersonal relations easily foster a schema altering environment. This is a limiting factor when developing software, because the environment becomes stale. The only solution for this 'staleness' is to have a continuous turnover rate. Hence, by keeping fresh people in the 'Cathedral' development model, the company is always allocated fresh ideas.

There *is* one flaw with the 'turnover' method used in the 'Cathedral' development process. When you have new people take over a project that was already started, acclimation and familiarization doesn't always occur. Some developers may implement functions that are not fully understood. Comment styles, and coding styles change drastically from person to person. The developers are not exactly familiar with how their portion of code fits into the project at large. This produces buggy and unstable software which winds up on most users' desktop. They take crashes in the OS, or software package, as a 'normal part' of computers, which is certainly *not* the case with open source software.

Open source software must be *usable*, and it must *work* or it goes by the wayside. The only way to achieve this is by having the source out in a type of environment that allows for maximum user and developer access. It must be placed in a public forum, agoria, where all eyes can evaluate and find bugs that are not so apparent to the developers, but that are apparent to someone attempting a particular task.

Divide and Conquer

When the task of developing the Linux kernel became overwhelming, the need for dividing it among members of the developer and user base became apparent. This form of modularization is another major advantage that the Open Source model has over the traditional 'Cathedral' form of development. Again, the

⁸Ibid.

key reason for the modularization of components in a software system is a way to maintain the stability of the final product and intensify development in particular software package areas.

This seems like it can transfer itself beautifully in a controlled closed source environment. However, by pure example (i.e.: Windows 9x/NT) , we see that this form of development in a closed software system detains the release of the software package. Why? As stated above, in a closed system there are some dedicated few developers, looking at the problem attempting to fix it, hacking at the source over and over again. They must be sure that the bug is fixed before the next release. This can and does delay the next release of the software package. If there are any more bugs released after a long hiatus, the user group will be even more disappointed.

With the Open Source model, division of the software package into key components becomes fully manageable. At the head of each division is a module developer; if the users have any problems with a particular module, they may scrutinise and find or fix a piece of code then inform the module owner of the bug and its fix. The module owner then knows how the fix is to be incorporated into the source tree and its local effects.

With Open Source you know exactly who the module owner is, and may communicate directly with him to exchange information regarding a problem or optimization specific to that particular module. This sort of personalized information exchange in the 'Cathedral' model would cause a massive level of confusion and disarray. Imagine FTP'ing a file in Internet Explorer and having it crash on you. Then imagine the number of people experiencing the same problems with Internet Explorer. Two problems would arise:

- ◆ In order to resolve the bug in the software, you would have to wait until a patch was released from Microsoft, due to its closed source policy. This could take from a few months to a few years.
- ◆ If the person's e-mail address who was working on that particular module were disclosed, and only 10% of the people who experienced the problem would e-mail the developer, his e-mail box would be flooded. Why? See previous point.

Many Open Source projects in development at the moment could not actually survive if it weren't for the modular design. The simple reason why these projects would go towards modular design is the distance involved between developers. Using the Internet as a common ground of communication between developers, yet being physically miles apart from each other, mandates that the code be modularized.

Free . . . What's the Catch?

Some people have offered criticisms against the Open Source paradigm. I will attempt to address the three popular 'catch' questions asked by individuals whom are unfamiliar with the Open Source paradigm.

Intellectual Property

The first question that many people have about Open Source, is that of copyright and intellectual property. This issue may be viewed in the following

way.

Intellectual ownership is maintained if *all* sources are Open, because then it would be quite apparent when someone has stolen an algorithm, or design, and not given proper credit where credit was due. While not all source is open, during the transition phase, the GNU Public License (GPL) may be attached to the code. This is the legal sword with which to fight battles should intellectual ownership rights be infringed by commercial and non-commercial closed source software developers.

Altruism or Naivete?

A second issue that might arise suspicion by critiques of the Open Software movement is that of 'coding for nothing.' They mock altruism as a quality of those who are naive. They state that people need a pecuniary reason to motivate themselves.

I offer this response as a counter argument to the question of reward.⁹

1. The pure joy of creation. Just as children enjoy making sand castles, or snow angles, adults love creating things of their own design.
2. People enjoy seeing others use what they have created and find the creation helpful to them.
3. A software project is very much like a clockwork device. It has pieces that interlock, interact and communicate with each other. All of these pieces were designed by their creator (the software developer) to behave as such. Hence when a software package is released, it gives full satisfaction to the parties involved to see their creation work elegantly, following all the rules. If the software project fails to follow the rules, then the team goes back and fixes that bug immediately, in order to attain this satisfaction.
4. With the development, and evolution of each software package, the designer, or team, and even the by-the-wayside on-looker perusing through k-loc's, learns something new at every turn. This learning cannot be as easily achieved (especially by the latter party) in a closed source system.
5. The developer also enjoys the cooperation of such a malleable medium as the computer. There are very few careers where such can be said. Much like the sculptor or the painter, the developer can take an idea and manifest it in a palpable form. He can build something out of nothing. As the sculptor materializes a likeness of some particular idea in his head to a tangible medium, so does the developer. It seems almost like magic to be able to build something with but a single step.
6. Necessity is the mother of invention. A developer will design a tool to fit a particular task. This tool is probably needed by others in the computing industry. Since the developer must design this tool to

⁹ These reasons were offered in the book "The Mythical Man-Month" as *Joys in the Craft* page 7. I am speaking as a code hacker and software systems developer for a major hospital in the Philadelphia area. They seem as valid reasons; I have adopted them as valid arguments for this issue in the Open Source movement.

begin with for his *personal or business use* there is no reason why he should keep it from the rest of the industry.¹⁰

Big business and the Open Source paradigm

And yet some mention, that corporations cannot make a profit in this manner. How would a corporation survive under the Open Source paradigm? Companies must learn, adapt to a society where Open Source is the norm. Currently companies thrive on the economic principle of scarcity. We see it everywhere in not only the computing industry, but also the entertainment industry, medical industry, engineering industry, automotive industry, etc.

The concept is quite blatant: If there is only one outlet for any particular product, then the consumers of the product are at the discretion of the monopolizing outlet. Hence, the originator of the product has full control as to the direction and monetary value of the product, regardless of the consumers' wants.

A prime example of this Microsoft Windows operating system. I would rather see a more stable OS than a 'prettier' one. Yet, they insist that 'they know what is best for me' as a developer and attempt to charge me US\$95 for the distribution. Microsoft controls the market based on scarcity. It is the only outlet with an x86 operating system that has the most hardware and software support, mainly due to their Rockefeller tactics.

The economic principle of abundance is the driving force for Open Source software. This principle is quite simple: The more you use and distribute a product, the more valuable this product becomes. Hence the customers are the actual driving force for the product. They dictate what direction the product should take.

Two prime examples are the Internet, and Linux. If the consumers of the Linux operating systems were not satisfied with the product, they would simply not use it. The product would die; Or the product would be driven toward the direction of excellence because *no one source has complete control over the direction or price of an open source product*¹¹ like Linux. Then it is quite apparent that Linux's value comes, not from its sale, but from its use. The more people use it, the more they rely on it, which increases its value. With increased value, people would be more prone to develop software solutions that would in turn ameliorate the upward spiral. Pecuniary interests may then be satiated by its use rather than its manifestation¹².

The same may be said about the Internet. The Internet's abundance (in this case, its omnipresence) for daily tasks such as e-mail, WWW, and general

¹⁰ He could possibly charge for the use of the tool. However, this creates certain problems within the closed source software industry. Each person or company within the industry must keep reinventing the wheel. This re-invention of the wheel produces a loss of productivity and an overall loss to the industry. See the Agorics papers article listed in the bibliography, particularly, the section on implementation of wide scale Agoric Systems.

¹¹ Linux Magazine, "Microscared: The Challenge of Open Source Software." page 26.

¹² For more information on the pay-per-use versus pay-per-copy issue see "The Agoric Papers" section 6.

communications, lends itself amicably to the economic principle of abundance. No *one* source has complete control over the Internet driving its cost and direction. It is, however, its own entity. Lest it be forgotten, the Internet started out as an Open Source venture by an “unprofessional” Internet Engineering Task Force (IETF). The Internet evolves, and takes on its own direction as the users deem it necessary for new protocols and standards to be developed. It adapts or it dies! And once again, pecuniary interests from this ‘free’ software project are satiated by its use and not by its presence.

Into the 21st Century

The computer industry is currently undergoing some major schema reorganizations. Open Source is finally, once again, peering its head from behind the shackles of multi-national corporations attempting to quell something which has gotten out of hand. End user companies, computer retailers, are now considering and packaging (respectively) the most famous implementation of the Open Source paradigm, Linux. For the first time, the Open Source paradigm is confronting these companies, like Microsoft and Sun Microsystems, on the only level where these companies are weak, a new economic principle. They must learn to adapt or they too will die; their tactics of spreading Fear, Uncertainty and Doubt (FUD) throughout the computing industry will only hasten their demise, as more and more people see through these underhanded tactics.

So here I offer this explanation of the Open Source paradigm in hopes that the contents would provoke thought among members of my community. This thought prodding is necessary at a time like this, while the networked computing industry is still at its infancy. All of this in hopes that the industry, of which I am a part, does not ‘shoot itself in the foot’ from mistakes past made, repeated.

“Perhaps the central problem we face in all of computer science is how we are to get to the situation where we build on top of the work of others rather than redoing so much of it in a trivially different way.” -R. W. Hamming, 1968¹³

¹³ Regarding the reinvention of the wheel each time the developer wants to impliment something for his particular project. From "One Man's View of Computer Science." ACM Turing Award Lectures: The First Twenty Years 1966-1985. p.216. See W.C. for more information.

Works Cited

- ▶ “Agorics Papers, The.” <<http://www.agorics.com/agorpapers.html>> May 23, 1999.
- ▶ Berger, Robert. “Microscared: The Challenge of Open Source Software.” Linux Magazine Spring 1999:27.
- ▶ Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Reading MA, 1995: pp. 3-15 and pp. 177-210.
- ▶ Hamming, R. W., "One Man's View of Computer Science." Ashenhurst, Robert L., and Graham, Susan. ACM Turing Award Lectures: The First Twenty Years 1966-1985. Addison-Wesley, Reading, MA, 1987: 216.
- ▶ Knudsen, Craig. “Troll Tech’s QPL.” Linux Journal. May 1999:86-87.
- ▶ “OSI Protocols.” <http://ken.slctech.org/miscdatacomm/osi_protocol.htm> May 23, 1999.
- ▶ Raymond, Eric. “The Cathedral and the Bazaar.” <<http://www.tuxedo.org/~esr/writings/cathedral-paper.html>> May 24, 1999.
- ▶ Zawinski, Jamie. “Resignation and Postmortem.” <<http://www.jwz.org/gruntle/nomo.html>> May 23, 1999.