A Dynamo for Computers: How Open Source Can Help Software Markets

Revision 2.1

Mattia Monga

August 29, 2000

1 Digital Physics

As the role of computers expands in every day life, the demand for software is increasing enormously. In 1998 in US about 2,400,000 of people was working as software specialist (1% of total US population) and a 700,000 units shortage was estimated [Jon99]. The need for software is much greater than its production and its quality is often far from what users would like to have.

How does industry cope with scarcity of goods? The recipe was suggested by Adam Smith in *"The Wealth of Nations":* a market regulated by property rules. In these days is generally accepted that the invisible hand of distributed interests is able to bring us plenty of goods on our tables[1]. In this view the economy of a product results from three factors:

1. the peculiarities of the product;
2. the infrastructures that build the market in which the product is exchanged;
3. the property rules that regulate the market.

Software is a digital good with the following peculiarities that affect its commercial properties:

- replication possible at no cost;
- ease of modification;
- marginal costs of production for modified artifacts near to zero.

[1] *"It is not from the benevolence of the butcher, the brewer, or the baker that we expect our dinner, but from their regard to their self-love, and never talk to them of our own necessities but of their advantages"*

Some researchers identify the principal cause of software crisis in an immaturity of its market [Bae98, Cox92, Cox95]. They suggest that software industry suffers from its product anomalies and they envision a software components market where components instances–rather than code–are rent, bought, sold and exchanged in the same way "hard" goods are [MK90, Cox96].

Traditional licences (as everybody knows, software is never really sold, but the producer give buyers a licence to use it) tend to limit the impact of these anomalies: replication is restricted to backup copies, modification is not allowed and hindered by hiding source code and distribution of modified copies is strictly prohibited.

In this paper we take a different approach: nor the software *anomalies,* neither the market structure is the cause of software scarcity, but we indicate the problem in property rules: what is effective and desirable for atoms, not necessarily is effective or desirable for bits too. We are going to show that licences that exploit software peculiarities getting over with (intellectual) property rules (*open source licences*) can produce more and better software, without changing the market infrastructures.

The model we have in mind resembles the one accepted for electro-magnetic induction, as synthesised in Moglen's Metaphorical Corollary to Faraday's Law[2]:

> Wrap the Internet around every person on the earth, then spin the planet, and software is going to flow in the network. [Mog99]

Software production is an essential creative activity, and like philosophy in the ancient Athens or arts in Italy during Renaissance, is fostered by:

1. a large community interconnected as much as possible (*wrapping*);
2. a way to give a wage to community members (*spinning*);
3. as few hurdles as possible in sharing knowledge and results (*resistance of conductors*).

If we want to produce as much software as our society requires, we have to build a metaphorical *dynamo* able to transform wages in software, leveraging on the Internet wiring. In this paper we try to explain why open source licences can increase the conductance of the Internet (Section 2) and how open source business models can pay programmers' work (Section 3).

2 Licensing Software

Software is not an artifact one can buy: the author sells users the right to *use* it, according to the restrictions expressed in a selling agreement called *licence.* The property rules regulating bits are licence agreements. In all western countries computer programs

[2]This is analogous to Faraday's law: *"The line integral of the electric flux around a closed curve is proportional to the instantaneous time rate of change of the magnetic flux through a surface bounded by that closed curve"* $\oint E = -\oint \frac{d\phi(B)}{dt}$ and its corollary known as Felici's law $i = -\frac{1}{R}\frac{d\phi(B)}{dt}$ where i is the electric current intensity and R the resistance of the medium

are protected by copyright laws[3]: in the European Union software is considered “literary work” and its protection is regulated by the “Berne Convention for the Protection of Literary and Artistic Works” and a specific Directive [eec91]. The Directive clarifies that the term ‘computer program’ (subjected to copyright protection) is to be intended as including the preparatory design material, but not the ideas and principles which underlie any element of it.

Licences regulate users’ acts with respect of licensed software. In particular:

Use: the loading, displaying, running, transmitting or storing of the computer program.

Copy: the permanent or temporary reproduction of a computer program by any means and in any form, in part or in whole.

Modification: the translation, adaption, arrangement and any other alteration of a computer program.

Distribution: any form of distribution to the public, including the rental, of the computer program.

Some national laws limits the restrictions that licences can pose (for example [eec91] states that back-up copies cannot be prohibited and modification for achieving interoperability with other programs is allowed), nevertheless this is an excerpt from Microsoft Windows 2000 Professional End-User Licence Agreement:

> You may permit a maximum of ten computers or other electronic devices to connect to the Workstation Computer to utilise the services of the Product solely for file and print services, Internet information services, and remote access (including connection sharing and telephony services).

As you can see the Product can be used *solely* in the way prescribed by the copyright holder. Furthermore, modifications are made nearly impossible by giving users only a binary form of the program: the design material, the source code and, often, even the format of output files are not distributed.

Hardware Market Licences, of course, could also permit users to copy, change and distribute the software and its intermediate artifacts. And indeed in the early days of computer industry, they were used to do so. Throughout 1960’s and 1970’s IBM distributed[4] its software accompanying it with source code and encouraging its customers to share improvements and adaptions. At that time IBM was the dominant computer manufacturer and it needed as much software as possible to sell its hardware.

[3]In US computer programs can also be patented. See `http://www.freepatents.org` for more information

[4]Initially the software was sold together with hardware, i.e., customers payed just for hardware (the only thing *real* engineers could buy) and the software was a free plus from IBM, but this policy was found unfair and US antitrust ordered the“unbundling” of the software, charging for it separately

Software Market With the advent of commodity hardware (PCs and cheap workstations) software market emancipated itself from the hardware one and novel software firms began to rest their incomes solely on the number of copies licensed. Releasing software is the mean to get returns on investments an even slightly modified versions of programs were released as new products: programmers work was primarily targeted to make applications perceived as new rather to create real new functionalities. Software providers began to cut the costs of interoperability and documentation, in favour of time to market, while at the same time excluding competitors. Users lost in freedom and quality, while developers as a community lost in productivity: each work was forced to reinvent the wheel[5].

Furthermore, most of developers were both users and programmers and they lost their ability both to change tools to improve their work and to increase the effectiveness of their products leveraging on users' improvements.

Free Software In 1984 Richard Stallman[6], a programmer of Massachusetts Institute of Technology (MIT) decided that the licence system was posing unacceptable limitations to his freedom of work and think. He founded *Free Software Foundation* and started the GNU[7] Project: an initiative to build a complete software system, from operating system to end-user applications, entirely distributed under the term of *GNU General Public Licence* (GPL) [Fou91]. This licence allows copy, modification and redistribution of the "human readable" version (i.e., the source code and all artifacts needed to build the "machine readable" binary form) of a program, on the condition that redistribution is made under the same terms of GPL itself. This condition, called **copyleft,** uses copyright rules to *prevent* the propriety of programs, resulting in software that is *free,* in the sense that it cannot have tyrants. It is worth noting that the GPL does not prohibit to sell licences, but allowing the existence of resellers, it limits the revenues from that activity. As we see in Section 3 profits have to exploited from other sources.

GNU software started with a core of utilities (an editor, a C compiler and a set of classical text processing tools) and became very popular among developers. Its popular-

[5]See analogies with the history of the formula for solving cubic equations. The first person able to algebraically solve cubic equations was Scipione Del Ferro around year 1510. However, the only person he told of his discovery was his student Fior. Fior was challenged by Niccolò Tartaglia to a debate about solving equations. Each was to write thirty questions for the other to solve. Fior submitted thirty questions of cubic type believing that his opponent would be unable to solve them. Tartaglia submitted a variety of questions of which Fior was able to work very few. However, inspiration hits Tartaglia the night of the debate on a method for solving cubic equations. With this new method he solved all of Fior's problems in less than two hours. Tartaglia was declared the winner. After many months and much convincing, Girolamo Cardano got Tartaglia to reveal his method for solving cubics. In 1545 Cardano published Tartaglia's rule for solving cubic equations in *Ars Magna*, much to Tartaglia's annoyance. Cardano gave Tartaglia full credit, but in those days, practitioners of mathematics got their fame by being able to solve problems no one else could, and if every reader of Cardano's work could solve a cubic, Niccolo's reputation would be worthless. Tartaglia spent the rest of his life trying to discredit Cardano [VdW85]. The effect for the community of mathematicians was that an important discover was delayed for thirty years.

[6]Richard Stallman is the creator of the well known text editor `emacs`.

[7]GNU's Not UNIX: see `http://www.gnu.org`.

ity was due mainly to its "portability"–i.e., its ability to be changed for using it on any platform–and its adaptability to users' needs. Two factors contributed to its worldwide spread and adoption:

1. the rise of a global cheap and permeating interconnection among its users (the Internet);
2. the apparition of some free kernels[8] (Linux and FreeBSD).

The existence of free kernels meant that it became possible to build a system solely based on free software and the diffusion of Internet allowed easy sharing of the improved versions of software.

Free Software Market Nowadays the GNU project has its own kernel (the Hurd), and estimated GNU/Linux[9] users are more than ten millions. The running infrastructure of the Internet (its mail transports, web servers, and FTP servers) are almost all free software, and there are free desktop managers, free applications for office and personal productivity, together with lots of tools for the developers' community. The role of software is increasingly an enabler for other computer activities [O'R99]. The free software concept begins to be popular outside the technicians' commonwealth. However, it is often misunderstood as software you can get at no cost. This is a dangerous misconception that can imply two equally erroneous beliefs:

1. *Software has negligible production costs:* every IT manager knows software costs are enormous, and usually its maintenance is even more costly.
2. *Free software people are hopeless utopians:* the effectiveness of free software projects[10] in building the Internet itself is a proof of concreteness of their ideas.

For coping with misunderstanding and the so called FUD[11], a new term for free software was forged: **open source software** (OSS). Proposers intend to have a word for talking about free software and its economic perspectives without scandalising managers. They give also a definition of the features of licences that can be considered free, i.e., open source. The Open Source Initiative (`http://www.opensource.org`) has now a certification mark, 'OSI Certified'. When the Open Source Initiative has approved the license under which a software product is issued, the software's provider is permitted to use the OSI Certified certification mark for that open source software.

[8]The kernel is the main part of an operating system.

[9]GNU system based on Linux kernel.

[10]It is worth noting that the major application protocols of the Internet are best implemented by free software programs: STMP (mail protocol) by Sendmail, DNS (domain names services) by BIND, HTTP (web protocol) by Apache

[11]Fear, Uncertainty, and Doubt: confusing arguments diffused by who has interest in conserving the *status quo.*

2.1 Open Source Licences

The Open Source Definition [Per99] specifies which liberties a licence must permit to qualify itself as OSS:

Use: no restrictions can be posed on the kind of users or fields of endeavour: the software must be usable by anyone for any end.

Copy: any number of copies of the software must be allowed without royalties or other fees.

Modification: anyone must be permitted to make derived works from the software: for this is key that distribution is accompanied by unobfuscated source code and all the scaffolding needed to produce the binary version.

Distribution: no restrictions can be posed on giving away copies and derived works, nor on the means used for such distribution. It must be always possible to license derived works under the same terms of the original one.

The Eulero-Venn diagrams of Figure 1 show that not all free downloadable software is really free software and a survey of compliance to Open Source Definition of the most popular licence models is summarised in Table 1. The table also shows if a licence gives a warranty that modified copies should be distributed under the same terms of originals: i.e., a free program cannot become not free (copyleft, see Section 2).

Table 1: Conformance of some licences to Open Source Definition

Licence	Use	Copy	Modification	Distribution	OSS	copyleft
public domain	free	free	free	free	yes	can be re-licensed by anyone
semi-free software	no-profit	no-profit	no-profit	no-profit	no	often
freeware or freely redistributable	free	free	no	not modified	no	no
shareware	free for limited time	free	no	no-profit	no	no
proprietary software	restricted	restricted	no	no	no	no
GNU General Public Licence (GPL)	free	free	free	free	yes	yes

continued on next page

continued from previous page

Licence	Use	Copy	Modification	Distribution	OSS	copyleft
GNU Lesser General Public Licence (LGPL)	free	free	free	free	yes	permits linking with non-free modules
Berkeley Source Distribution (BSD), MIT X Consortium, and Apache Licences	free	free	free	BSD and Apache require that every advertisement mentions original developers	yes	no
Artistic	free	free	free	not clear: the single program cannot be sold, but aggregates can	yes	no
Netscape Public Licence (NPL)	free	free	free	free	no	licencers can use contributed code in their proprietary versions
Mozilla Public Licence (MPL)	free	free	free	free	yes	weak
Qt Public Licence (QPL)	free	free	only by patches	free	yes	no
IBM Public Licence	free	free	free	free	yes	yes

continued on next page

continued from previous page

Licence	Use	Copy	Modification	Distribution	OSS	copyleft
Sun Community Source License	free	free	free	no	no	no
Apple Public Source License	free	free	free	free, but can be revoked by licencers	no	no

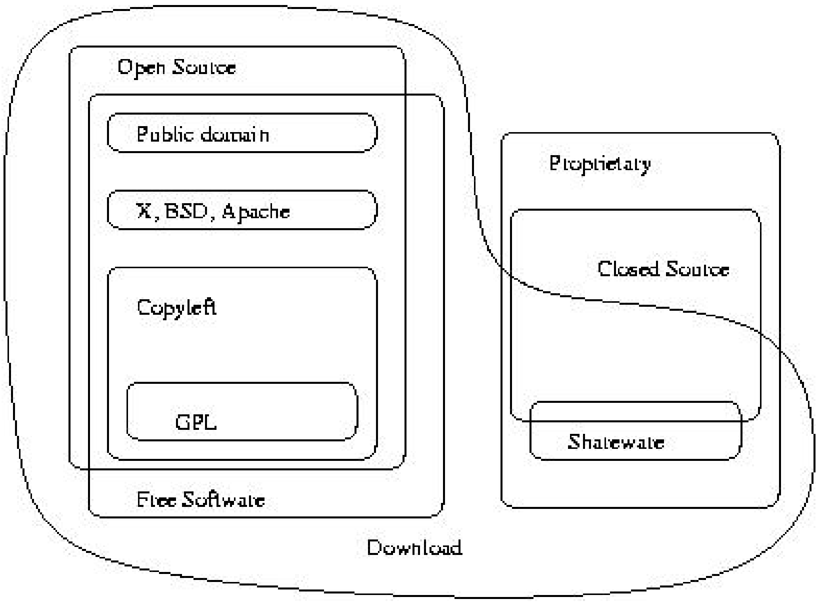

Figure 1: Software categories

2.2 Open Source Development

Fulfilling the demand for software of our society is a difficult task. Software development is a creative activity, inherently complex, costly, and time consuming. Often, as noted by Fred Brooks [Bro95], men and work-time are not interchangeable: you cannot simply add more programmers to a project to end it earlier. This is because the cost of the coordination of the team increases with the square of the number of involved people, while effort profused can increase just linearly. The gestation of a baby lasts nine months, even if two women collaborate to it. But a network of friends can greatly alleviate mother's problems. Opening the source code enable the grow of an helping community that can decrease the costs of software production, redistributing them among more principals. In this section we show how actors can take advantage of the peculiarities of the open source development model during the process of software release.

2.2.1 Skills Detection and Project Management

A serious problem project managers has to face with is hiring good and motivated programmers. People who externally participate to an open source project (that is users that improve and adapt code) is skilled ‘by construction’. In fact, this can even be a problem: only skilled people can contribute to development. However, the skills required are very different: all users become beta-testers, some can do customisation, most talented ones act as co-developers.

The real problem is the cost of coordination: according to Brooks’ Law [Bro95] these are proportional to the square of the number of involved people, however, there is an important difference with respect to proprietary software production: they do not burden only the core team, but they are distributed among all interested parties. Open source development teams are generally composed by a large set of contributors with a smaller set of core individuals. They often resemble (except for scale) the “surgical” teams envisioned in [Bro95], as an extremely productive organisation.

2.2.2 Requirements

Every good work of software starts by scratching a developer’s personal itch [Ray98]. This is often the case of free software programs: they start with a core set of features needed by the first “buyer” and expand to scratch all itches of people in the same problem space. The advantages of this approach are twofold: the development become naturally incremental, teams can *grow* much more complex entities than they can *build* [Bro95]. Furthermore added features are localised and engineered by their real effectiveness: no theoretically use cases focus, but real users drive the development and single features are often autonomous with respect of core program. Requirements and features are kept consistent by a release-feedback cycle that can be an order of magnitude faster than proprietary one.

The effect of open source on effectiveness of software results particularly evident in environments where users are hard to satisfy. Even when original developers lose interest in their software, the program can survive (i.e. can still “scratch their emerging itches”) if an adequate mass of users continue to exist. Starting an open source project can be very difficult, but if you are able to build a flywheel of collaborating users, benefits can be expected.

2.2.3 System Analysis, Design, and Implementation

The main goal of system architects of an open source program is to simplify the structure of the software as much as possible. The success of the product is tightly coupled to its design elegance. The end goal is always to obtain a network of collaborative users as wide as possible: comprehensibility, uniformity, modularity, changeability are all key factors for program diffusion. Linux kernel and Emacs editor, for instance, are so ubiquitous thanks to their clever design: they can be easily customised and expanded understanding a little of their internal intricacies because of their component approach; their code (not really a trivial one, Linux consists in about 10 millions of lines of C code) is easy to study, modify, port to different platforms and also re-engineer: Linux Real Time and XEmacs are branches of evolution heavily re-designed.

Furthermore, exploration of design space can be carried on in parallel by different, possibly competing, teams and only successful solutions reach the code base. Again, the problem can be coordination, that is the ability to get the right information to the right people at the right time [HGM98]: mailing lists are the main channel of communication, but in a big project the retrieval and management of information can be heavy and lot of opportunities could be lost.

2.2.4 Test

"Giving enough eyeballs all bugs are shallow" [Ray98]. This over-cited sentence summarise the Linus[12] law: debugging is a parallelisable activity and require low costs of coordination. Users become a legion of beta-testers that can not just *diagnose* problems, but also *resolve* them. Again, Linux kernel is an instructive example: dozens of security bugs in its network modules were resolved a few hours after someone discovered them, and now it is known as one of the most secure kernel on the market. According to [MKL$^+$98] the failure[13] rate of utilities on the proprietary versions of UNIX ranges from 15% to 43%, while the failure rate for Linux is 9% and for the GNU utilities is only 6%. Interestly enough, the proprietary versions of UNIX do not correct all the bugs[14] reported in a eight years older analogous study [MFS90]: unfortunately that report did not contain data about Linux nor GNU utilities. Open source programs in the long run can be much stabler than their proprietary counterparts, but, it is important to note, that testing is not just debugging, and formalised procedures of testing are unaffordable by the open source community in the large and have to be relegated to single users (or ad hoc firms, see Section 3), because of, again, coordination costs.

2.2.5 Commercial Distribution

In an open source world there are much less barriers between developers and end-users. Acquisition costs are normally very low and often a lot of documentation is available. Software houses often distribute even their proprietary as shareware to take advantage of the same benefits.

[12]Named after Linus Torvalds, the creator of Linux kernel.

[13]In the paper "failure" means a crashing with core dump or hanging (looping indefinitely)

[14]Network utilities are exceptional regard to this: in [MFS90] study only two programs crashed, no one crashed in [MKL$^+$98] study. The power of interconnection in improving quality of software seems evident.

3 Open Source Markets

As noted in Section 2, open source licences do not prohibit to *sell* the right to use, copy, modify, and redistribute, but giving up with the monopoly allowed by traditional licences, such sale is rarely profitable and development costs of open source software cannot be paid back by its sale value. But software has an enormous economic value as a tool: it embodies knowledge of how to accomplish some purpose [Bae98]. Therefore, it is feasible exploiting software *use value*–instead of sale value–to generate programmers wages, leveraging on a continuing exchange of value between vendor and customers. Indeed, only sale value is decreased by abandoning the proprietary software model: use value can even increase, because widespread use fosters standards adoption, bug fixes, and addition of new features. New business models funded by software use value were exploited [Ray99] by open source enterprises.

3.1 Indirect Sale Value

Some of them raise profits from indirect sale value. Rather than bits, this kind of firms sells:

Services: The core assets are the brains and skills of employees Their ability to use and "customize" open source software is provided to solve customers problems. The added value of assembling, testing, customizing a program is sold together with open source software and the main part of customers may evaluate more efficient to hire experts than to develop in-house skills. This is the model adopted also by lawyers[15]: national laws are free and accessible by everyone, but they provide expertise and ability to adapt general rules to individual cases, and people pay for these services. The exchange of value between the business company and customers is continuous, because it is related to the real *use* of the software: as long as the software is used it should be kept in synchronization with user needs and the value of such synchronization it will be available to be shared between the customizer and the customer. Red Hat Inc. (`http://www.redhat.com`) has the largest market share of GNU/Linux CDs. Although the same CDs are sold by CheapBytes at $1.99, RedHat can sell its ones at $49.95 because they give to customers also 90 days of on-line support. Many traditional business models for proprietary software can be applied thanks to the high satisfaction of customers with respect of packaged products (no customization at all can be sold, if the product satisfies customers perfectly), and monopolies force customer satisfaction.

Accessories: Often, getting the software itself is not enough. Computers, printed manuals, training courses, skill certifications and even gadgets[16] are all atoms that could be bought by bits users. The more the software is freely available, the more the users will pay for accessories.

[15]As noted by Eric Raymond, typically, they are not doomed to starvation

[16]`http://copyleft.net` sells T-shirts and merchandise related to open source programs

Contents: Software can be a mean for enjoying contents: online services, but also multimedia CD, etc.. If the enabler program is free it is likely to be ported and enhanced to new platforms by someone willing to use it on her machine and the market of contents may expand.

Brand: Drinking water flows out from our taps at negligible costs. Nevertheless we often buy *branded* water, sold in bottles. `http://counter.li.org/` keeps a record of GNU/Linux users that want to register themselves. Of the current (May 2000) 143956 registered users 24.4% got it from the net, 43.75% bought a CD with a brand on it, and the rest got it by other means (friends, magazines, etc.). The added value of having a well known set of programs and someone that has tested them together is recognized and payed by users. A firm can also sell certifications of compliance to branded standards.

Hardware: Hardware companies have often to write and maintain software needed to operate their products. Device drivers and configuration tools for printers, peripheral boards, etc. are not profit centers, but rather cost centers. Opening the software can alleviate maintenance and porting costs and increase interoperability among different systems.

One might argue that these business models are practicable also with proprietary licenses. In fact this is not entirely true:

- customization and services for packaged software become profitable in quasi monopolized markets: Microsoft and SAP can play in these markets because of their market share.
- Training courses and certification programs are not well accepted when knowledge is perceived as proprietary. What did they learn if their work environment is about to change?
- Quality of proprietary software is certified by owner itself: why people should trust certifications? Open source software quality can be assured by third party companies.

Therefore, although practicable, these business models appear to be much more profitable if the software is distributed freely.

3.2 Other Business Models

Sometimes companies whose core business is different from software can find useful to use open source licenses for tactical purposes:

Platform: Netscape Communications Inc. revenues are mainly from services related to its server-side software. This business was threatened by Microsoft, that could drive out Netscape changing client-side rules of the game. In 1998 Netscape effectively denied Microsoft the possibility of a web browser monopoly releasing its browser to open source community. This model (called loss leader model in [Ray99]) seems to be the best comprehended open source model and several traditional software houses have adopted it for some of their product in the last months.

Cost sharing: When a project is critical for complexity and reliability, open source may pool efforts into improving a common code base: parallel development efforts and massive parallel peer review can increase enormously code effectiveness. Scientific research leverage on the same model to obtain better and better theories.

Risk spreading: Open source software never dies. A closed program dies with its producer. The source code of an open source program is released to the public and until there is an user, it is feasible to get a programmer working on it, maintaining it, customising it, improving it. Users can spend more for an open source program because they reduce the risks attached to their investments[17].

Process benefits As stated in 2.2 open source licenses can be beneficial on the software production cycle refining requirements, improving involved skills, modular design and quality of testing procedures, but on the other hand it is necessary to cope with increasing coordination costs.

3.3 Open Source Models Essence

If, as stated in [Por93], customer satisfaction is the main requirement for survival of firms, open source software must eventually increase the value of a company in the software market, because users definitely benefit from free software peculiarities. However, it is obvious that different actors of the software market benefit differently from the introduction of open source licenses. We try to summarize different effects in the following table:

From Table 3.3 appears clearly that the introduction of open source licenses might damage the revenues of software houses, because they lose their role of intermediation in aggregate value for developers. Simple users, hackers and developers tend to form a community on their own, skipping intermediation, and software firms have to find a new role, producing much more added value to be profitable. Figure 2 shows value exchange in the case of a traditional proprietary software house (without a monopolistic position in the market) and an open source company like Red Hat. The money exchange between customers and the business firm is aggregated in *space* in the first case and in *time* in the second. Developers are remunerated by money, but also by *fame* in the case of open

[17]Let me describe a little personal experience: I was using an open source e-mail client (TkRat) and I was not happy because new messages were not easily distinguished from old ones. So I decided to spend a couple of hours looking in the source code for a way of coloring message headers according flag criteria. After some work I was able to modify the program according to my desires. But this posed a problem: for any new version of the program released by the original author, I needed to add my features. So I decided to return my patches to the author, hoping he incorporated them in the code base. He did, and now those featured are likely to be maintained by the open source community forever.

Actor	Advantages	Disadvantage
Customer without any hacking skill	freedom of use freedom of copy freedom of distribution freedom from software producers common standards program are more robust security cannot be based on obscurity	Are there any disadvantages?
Hacker	All the advantages of ordinary users freedom of change source code more accurate documentation personal satisfaction they can work as free-lances	none
Traditional developer (payed by a software firm)	design elegance less pressing deadlines	increasing costs of coordination
Software house	deeper testing widening of the market easier skills detection spreading of maintenance risks	loosing of monopoly constraints no profits from sell value less barriers on competitors entry change of business model

Table 2: Pro et contra

source. In the open source case the market is much more tightly coupled: the software tend to be the tangle that weaves together market actors. The more the software is an *infra-structural* one, the it can be the conductor for value exchanges; the more the software is inert (think about a *perfect* end-user application), the more the software house can ask for money for its–proprietary–role of conveyer of value. This also explains way open source licenses began to raise profits with the diffusion of the Internet: much more software becomes an infra-structural one in a world wide network environment.

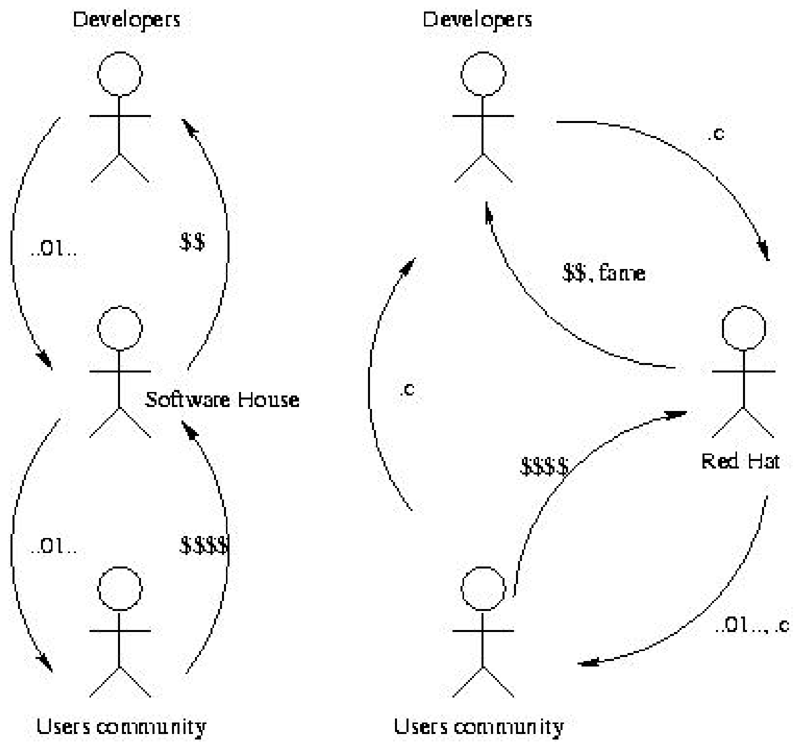

Figure 2: Value exchanges in an open source software market

4 Conclusions

Satisfying the demand for software of our society is a challenging task. We need reliable software that help us in solving our everyday problems. Software production is overwhelming complex and it is not easily increased simply adding new labour power. Traditional licences, based on monopolistic exploiting of intellectual property rules, hinder the interconnection between developers' and users' skills. Instead, with open source licenses (see Section 2.1) customer satisfaction is reached by increasing reliability and matching to users' needs, leveraging mutual collaboration among core programmers and hacking users. Developers compensate the loss of customer control with benefits gained along the way of software release (see Section 2.2. Software houses do not need to be charity organisations, they just have to get their profits also from different assets (see Section 3). Nothing new: sharing feedback and cross-fertilization among research, professional and didactic activities is the traditionally way knowledge is exchanged. Software revolution just showed that knowledge can be digitally packed.

References

[Bae98] Howard Baetjer. *Software as capital: an economic perspective on software engineering.* The Institute of Electrical and Electronics Engineers, Inc., I edition, 1998.

[Bro95] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley publishing company, USA, anniversary edition, 1995.

[Cox92] Brad Cox. What if there is a silver bullet and the competition gets it first? *Journal of Object-Oriented Programming*, June 1992.

[Cox95] Brad Cox. "No silver bullet" reconsidered. *American Programmer*, November 1995.

[Cox96] Brad Cox. *Superdistribution: Object as Property on the Electronic Frontier.* Addison Wesley, Reading, Mass., 1996.

[eec91] Council directive 91/250/eec of 14 may 1991 on the legal protection of computer programs. Official Journal of the Council of European Communities L122, may 1991. Community legislation in force.

[Fou91] Free Software Foundation. Gnu general public license. http://www.gnu.org/copyleft/gpl.htm, June 1991.

[HGM98] H. Holz, S. Goldmann, and F. Maurer. Working group report on coordinating distributed software development. In *Proceedings of 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'98).* IEEE Press, April 1998.

[Jon99] Copers Jones. The Euro, Y2K, and the US software labor shortage. *IEEE Software*, May/June 1999.

[MFS90] Barton Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. ftp://grilled.cs.wisc.edu/technical_papers/fuzz.pdf, 1990.

[MK90] Ryoichi Mori and Masaji Kawahara. Superdistribution: The concept and th architecture. *The Transactions of IEICE*, (73), July 1990.

[MKL$^+$98] Barton Miller, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl. Fuzz revisited: A re-examination of the reliability of unix utilities and services. ftp://grilled.cs.wisc.edu/technical_papers/fuzz-revisited.pdf, February 1998.

[Mog99] Eben Moglen. Anarchism triumphant: Free software and the death of copyright. http://emoglen.law.columbia.edu/my_pubs/anarchism.html, August 1999.

[O'R99] Tim O'Reilly. Hardware, software, and infoware. In Chris DiBona, Sam Ockman, and Mark Stone, editors, *Open Sources: Voices of the Open Source Revolution*, pages 189–196. O'Reilly & Associates, Sebastopol, CA, first edition, January 1999.

[Per99] Bruce Perens. The open source definition. In Chris DiBona, Sam Ockman, and Mark Stone, editors, *Open Sources: Voices of the Open Source Revolution*, pages 171–188. O'Reilly & Associates, Sebastopol, CA, January 1999.

[Por93] Michael E. Porter. *Il vantaggio competitivo.* Edizioni comunità, Milano, iv edition, 1993. Original Title: Competitive advantage.

[Ray98] Eric S. Raymond. The cathredal and the bazaar. http://www.tuxedo.org/ esr/writings/cathedral-bazaar/, November 1998.

[Ray99] Eric S. Raymond. The magic cauldron. http://www.tuxedo.org/ esr/writings/magic-cauldron/, June 1999.

[VdW85] B.L. Van der Waerden. *A History of Algebra.* Springer-Verlag, New York, 1985.