Why Open Source software can succeed[+]

Andrea Bonaccorsi[a, *], Cristina Rossi[a, b]

[a]*Laboratory of Economics and Management, Sant'Anna School of Advanced Studies, P.zza Martiri per la Libertà 33, 56127 Pisa, Italy*

Forthcoming in Research Policy Special Issue on Open Source Software Development (2003)

Abstract

The paper discusses three key economic problems raised by the emergence of Open source: motivation, co-ordination, and diffusion.

First, the movement took off through the activity of a community that did not follow profit motivations. Second, a hierarchical co-ordination emerged without proprietary rights. Third, Open Source systems diffused in environments dominated by proprietary standards.

The paper shows that recent developments in the theory of diffusion of technologies with network externality may help to explain these phenomena. A simulation model based on heterogeneous agents is developed in order to identify the relevant factors in the diffusion of the technology.

[+]A preliminary version of this paper was presented at the Conference *Open Source Software*, Pisa, December 2000 and at the AICA Conference, Bari, September 2002. Two anonymous referees and the editors gave precious comments, which greatly improved the paper. Jean Michel Dalle, Nicolas Jullien, Paola Giuri, Mauro Leoncini, Giovanni Santini and Guido Scorza also provided detailed comments. We acknowledge the contribution of Fabio Vistori in the preparation of the simulation exercise. We remain responsible for the content of the paper.

[*]Corresponding author. Andrea Bonaccorsi, address: P.zza Martiri per la Libertà 33, tel.: +39-050883323; fax: +39-050883344, e-mail address: bonaccorsi@sssup.it.

[b]Tel.: +39-050883343; fax: +39-050883344, e-mail address: cristina.rossi@iit.cnr.it (Cristina Rossi).

Keywords: Open Source, Diffusion, Network Externality

1. Introduction

From an economic point of view Open Source software can be analysed as a *process innovation*: a new and revolutionary process of producing software based on unconstrained access to source code as opposed to the traditional closed and property-based approach of the commercial world. This phenomenon raises a series of important questions that one, for reasons of simplicity, might formulate in terms of Schumpeter's three-phase axis of the innovative process, i.e. invention, innovation and diffusion (Schumpeter, 1934). For each of these phases the Open Source phenomenon generates extremely interesting theoretical puzzles for economic analysis, which can be formulated into the following questions:

- Why do programmers write Open Source codes if no one pays them to do it?
- How do hundreds of people dotted around the world manage to effectively co-ordinate with each other in the absence of any hierarchical structure based on the ownership of assets, producing programmes consisting of millions of lines of code?[1]
- Why is it that Open Source programmes are becoming so widespread in a world dominated by Microsoft-imposed standards, and more generally by proprietary standards?

The first question regards the invention phase: there is no problem in explaining an individual inventing the concept of free software and proposing it to the programming community, but that still leaves unexplained the phenomenon of hundreds of individuals continually inventing programmes which are then made available free of charge. What system of incentives regulates the initial innovation phase? What motivates such behaviour?

The second question concerns the problem of co-ordination, which is crucial for the transformation of an invention into economically advantageous and marketable innovations.

[1] According to J. Sanders (1998), in 1998 the kernel of the Linux operating system comprised over one and a half million lines of code. In the first version, implemented by Linus Torvalds in 1991, there were 10,000 lines.

The spontaneous, decentralised functioning of the Open Source community represents a challenge to many current notions of co-ordination. How is it possible to align the incentives of several different individuals without resorting to property rights and related contracts? How is it possible to specify everyone's tasks without resorting to a *formal* hierarchical organisation based on ownership of assets?

Finally, the third question focuses on the conditions for the diffusion of Open Source software in the population of users. How is it possible to increase the diffusion of new technology given the presence of a well-established standard, and thus of increasing returns from the adoption of the previous technology?

The aim of this paper is to explain what at first sight would seem "non sense" by combining recent developments in economic analysis such as the theory of collective action and the theory of technological diffusion in the presence of network externalities. It will become clear how a non-conventional understanding of the phenomena can provide a plausible explanation.

2. Motivational problems in the invention stage: Open Source software and collective action

Stallman proposed a revolutionary idea in 1984 with the Free Software Foundation, subsequently confirmed in 1997 in the Open Source Definition[2]. The key concept is that there should be unrestricted access to computer programming codes: anyone should be able to use them, modify them and circulate such modifications without having to pay anything.

The basic Open Source software philosophy is extremely simple: when programmers are allowed to work freely on the source code of a programme, this will inevitably be improved because collaboration helps to correct errors and enables adaptation to different needs and hardware platforms[3]. This has in fact happened and Open Source software is well known today

[2] Open Source Definition, http://www.opensource.org

[3] Introduction to Open Source, http://www.opensource.org

for its high degree of reliability[4] and portability. In the new approach, the hierarchically organised and top down planned structure adopted in all productive processes is abandoned in favour of a new kind of bottom up structure, which is non-coercive and largely decentralised.

The first urgent question is therefore the one clearly put by Robert Glass (1999, 104): "I don't know who these crazy people are who want to write, read and even revise all that code without being paid anything for it at all". To understand how this is possible, it is necessary to refer to a characteristic feature of the vast mass of programme users – their heterogeneity. They are extremely diverse in terms of know-how, interests and user needs, forming a number of sub-groups with quite different profiles. First of all, there is a large group of individuals who are not capable of developing programmes but only of using them, and only then if they are decidedly user-friendly. Clearly these people will never write free code, especially as available commercial software responds very adequately to their needs. Second, those who do write it are people for whom the computer is a hobby, programmers who spend hours in front of a screen in their spare time and at weekends. Their contribution is inevitably limited; hobbyist forms of motivation are clearly insufficient to explain the enormous results achieved by the Open Source movement. To do so, it is necessary to refer to a third group, formed by the members of the *hacker culture* – the heirs of the "Real Programmers" of the immediate post-war period, who "came from the engineering and physics fields…[and] programmed in FORTRAN and another half-dozen languages that have now been forgotten" (Raymond 1999b, 19). Although at the origins of computer revolution neither large universities nor large companies readily accepted computer science and software as respected disciplines, several generations of computer scientists were generated in the academic world and the research centres of large software corporations.

[4] The Web server equipped with the Linux Red Hat 4.2 operating system and Apache, installed in 1997 by Russell Pavlicek (1999) in the laboratories of the Compaq Computer Corporation, resisted 267 times before crashing and even then it only did so following a massive storm that led to a total power cut in the

In order to understand the Open Source invention phase it is necessary to analyse the motivation underlying the decisions of these individuals, who choose quite spontaneously to produce code lines. It should be emphasised that altruism might at most explain the behaviour of people writing software in their spare time but not the behaviour of those who have devoted considerable resources of time and intellect. But if it is not altruism, why work for free?

First of all, the production of Open Source software is a form of intellectual gratification with an *intrinsic utility* similar to that of a scientific discovery, involving elements other than financial remuneration (Perkins, 1999, 81). Emerging as it does from the university and research environment, the movement adopts the motivations of scientific research, transferring them into the production of technologies that have a potential commercial value. Sharing results enables researchers both to improve their results through feedback from other members of the scientific community and to gain recognition and hence prestige for their work. The same thing happens when source code is shared: other members of the group provide feedback that helps to perfect it, while the fact that the results are clearly visible to everyone confers a degree of prestige which expands in proportion to the size of the community[5].

Secondly, besides being a form of intellectual work, hackers also regard programming as an *art form*. Several authors describe Open Source programming as artistic satisfaction associated to solving complex computer problems[6].

Thirdly, in the new paradigm of development, programmers frequently rediscover the *pleasure of creativity*, which is being progressively lost in the commercial world, where the nightmare of

---

building where it was housed. According to Liebovitch (1999, 42) in comparison to Windows NT, Linux has a resistance to crashing that can be measured in "months and years rather than days or weeks".

[5] Within the increasingly vast hacker community the qualities that are appreciated are "intelligence, knowledge, technical ability, personal productivity, dedication" (Perkins, 1999), also because programming is "love not money" (Sanders, 1998).

[6] As an example, computer engineer Ellen Ullman (1998), author of the book *Close to the Machine*, writes in emphatic tones about Linux being a reaffirmation of our human dignity and describes her first installation of the operating system as "an exhilarating succession of problem-solving challenges" (Sanders, 1998).

delivery deadlines is transforming production into an assembly line, brilliantly described in *Microserfs* (Coupland, 1996), the cult book about young workers at Microsoft. "Microsoft-phobia"(Dalle and Jullien, 1999, 14) is undoubtedly something shared by all members of the Open Source movement. The reasons for this are not purely ideological but also, indeed above all, *technical*. Commercial software is primarily perceived as not being very reliable. Produced by a restricted group of programmers in obedience to market laws, it is in diametric opposition to the law expressed by Raymond (1999a): "Given enough eyeballs, all bugs are shallow".

Intellectual gratification, aesthetic sense and informal workstyle are all recurrent features of the set of different motivations underlying the invention of Open Source. In economic terms, these motivations refer to *intrinsic* or *non-pecuniary* rewards. But they are not sufficient to explain the persistence of the behaviour of programmers, because a self-interested form of behaviour would end up fatally compromising the mechanism of free software production. In fact this has *not* happened, and there are no signs that the original motivation has degenerated. In fact, there are other more conventional motivations to be considered.

One element is that working on Open Source projects provides the prestige and visibility that often gives programmers the chance to be noticed by software firms[7]. This is the main motivation emphasised in Lerner and Tirole (1999): working freely for the Open Source movement would be an investment activity aimed at increasing the *signalling of quality* of human capital.

Another element is that many Open Source projects take shape because the people promoting them have looked in vain for a programme to perform a particular function. They arise, that is, to satisfy a demand for which there is no corresponding supply, in short to "fill an unfilled market" (Green, 1999)[8]. In some sense the economic rationale here would be *self-production*.

[7] As Di Bona, Ockman and Stone note (1999, 14), all of the founding fathers of the movement "have earned enough of a reputation to allow them to pay the rent and maintain their kids".

[8] One typical example is the Perl language developed by Wall in 1987. According to Wall, Perl originated because of an unsatisfied need. In his work as systems administrator at Burroughs, he saw a need for

It is then possible to give an economically sensible response to the first question. Stallman and the other founding fathers of the Open Source movement have brought about a profound change in the nature of the software good, which has actually assumed the characteristics of a *collective good*. Although it is well known that free riding does not take place systematically in the real world, in this specific case what still needs to be explained is how self-interested, free riding behaviour is avoided and how sustained co-operation is also maintained *in such a large-scale community*. In order to give a final answer to this question, we need to analyse how the production of Open Source software is co-ordinated and governed. This analysis will show that, contrary to most free riding problems, in this case the incentive to contribute to a collective good does not get smaller the larger is the number of other contributors.

3. The governance structure of Open Source: communication protocols, software complexity and the role of licence agreements

Several studies (Ghosh and Prakash, 2001; Schmitz 2001) report about 12.000 active Open Source projects. Given that, on average, about twenty programmers work on each project and that each programmer contributes on average to two projects, we obtain an estimate of a community of 120.000 developers. Such developers have supplied more than 1000 megabyte of code, which amount to around 25 million lines of code (LOCs).

The number of projects posted on the Internet increases steadily. Some of these projects are fated to die but others are getting better and better. Among the most cited (besides the Linux operating system and the Web server Apache) there are BIND, that performs DNS resolution

---

something combining the speed and immediacy of the Unix shell languages with the possibility to develop more complex programmes, a typical feature of high-level languages like C. Unable to find that "something", he created and then distributed it to the entire Open Source community. This provided a number of clear advantages and today Perl is a commonly accepted language used above all for Internet-related applications. The same thing happened for Ethernet drivers developed by NASA's Donald Becker, who needed faster drivers for a project he was working on (Raymond, 1999e).

and Sendmail for the exchange of electronic mail. These two programs were the killer applications in their market: no commercial competitor succeeded (or wanted) to attack their supremacy.

In order to understand the governance structure of such a massive and pervasive phenomenon, the Open Source projects have to be the units of analysis. Open Source projects in fact not only perform the co-ordination task of hundreds of developers scattered around the globe but they also represent the institution in which the whole Open Source production paradigm becomes effective (Hecker, 1999, 50).

Looking at the wide variety of the Open Source projects posted on the Net, it is possible to identify some shared characteristics dealing with the project lifecycle, the network of the relations among its members, the linkages with the Open Source community and the proprietary closed source software.

An Open Source project is defined as "any group of people developing software and providing their results to the public under an Open Source license" (Evers, 2000). The group of people working on the project and the licence under which the source code is released are the key elements of the definition. A successful project typically starts because an individual (or an organisation) is facing a problem that needs a particular kind of software in order to be solved. Looking hints within her social network, she finds out other agents sharing the same concern. They form the embryonic informal group that starts working on the problem. The group eventually consolidates placing the project on a sound basis. As the project goes on, the group looks for feedback making the results publicly available on the Net through mailing lists, newsgroups or other on line news services. In this way the project attracts further participation that brings new tangible and intangible resources, while programmers that are not interested on the project drop away. The increase in participation allows the project to gain momentum, frequent releases lead to bugs correction and generate a feedback flow that improve the code and the understanding of the underlying problem.

Two factors shape the lifecycle of a successful Open Source project: a widely accepted leadership setting the project guidelines and driving the decision process, and an effective co-ordination mechanism among the developers based on shared communication protocols.

Most successful Open Source projects, far from being anarchical communities, display a clear hierarchical organisation. On the other hand, contrary to what happens in the proprietary paradigm, the roles within the hierarchy are not strictly assigned since the beginning. As the project grows, the authority of the project leaders (core development group) arises naturally from a bottom up investiture as a result of the contributions to the commonly agreed goal. Such a structure often leads to a highly concentrated distribution of contributions[9].

The leadership deeply influences the evolution of the project by selecting the best fitting solution among the ones that different contributors propose for the same problem. On the other hand the core development group does *not* carry out the bulk of the co-ordination effort. In general, no one in the project is forced to perform a particular task but agents choose freely to focus on problems that they think to best fit their own interests and capabilities. In presence of a pending job the strategy that is commonly adopted lies in waiting for someone that volunteers his contribution. When no one volunteers the job is neglected. Clearly if such a job is fundamental for the continuation, the project is going to stop[10]. In such cases the core development group may ask for contributions through the project mailing list or Web page, but no coercive mechanism supports such announcements. Nevertheless, in successful projects, the pending tasks are carried out by more than an individual or an organisation. Redundancy

[9] Referring to the active Open Source projects, it is estimated (Ghosh, Prakash, 2001) that 10% of the authors write more than 70% of the produced code, with the first 10 developers producing over 25% of total code. Moreover, productivity varies across authors. Only 25 programmers take part to more than 25 projects while the large majority contributes only to a single project. According to Mock et al. (2000), the 15 top developers of Apache added almost 90% of the LOCs and Koch and Schneider (2002) found that, in the GNOME project, 52 programmers out of 301 wrote 80% of the code.

[10] In any case, the source code continues to be publicly available: another group of developers can download it, perform the pending task and restart the project.

improves the quality of the code allowing to choose among a wider set of solutions. An emergent property of this system is therefore selective capability.

The co-ordination mechanism shared by developers is in most cases able to produce a well-structured flow of contributions. As happens for project leadership, co-ordination among developers is solved according to a bottom up approach that relies on generally accepted protocols of communication and a shared notion of validity. The strength of these elements is enhanced by the peculiar characteristics of the software good and the by structure of the Open Source licence.

Software is a complex product not only because it is a medium to solve complex problems but also because it results from the different features that can be combined in various ways (Franck and Jungwirth, 2001). Such complexity has deep consequences on the quality of the software product and on the efficiency of its production process. The development of object oriented programming represents a fundamental process innovation in the way of managing such complexity. In the object-oriented framework, a program is no longer a monolithic entity, but is organised in several modules. Such modules can be combined in different ways and reused in other programs. So the same modules may allow the solutions of very different problems making effective the concepts of modularity and reusability (Baldwin and Clark, 2002; Bonaccorsi and Rossetto, 1999). This fits very well the characteristics of the Open Source production process. Inside each program, the combination of modules takes place according to a structured hierarchy of dependence relations, but modules entering at the same level of the system can be developed independently from each other. Given that, the division of labour among programmers inside a project may be simply performed allowing different programmers to work on independent modules that are finally combined to form the whole structure.

According to its inventor, the key to the success of the Linux Open Source is its modularity (Torvalds, 1999). The most important feature of Linux is, in fact, its small and

compact[11] kernel that carries out the basic operating system tasks. The system capabilities can be extended by independent modules that are loaded on demand when the system is active and particular jobs need to be performed and can be automatically unloaded if they are no longer required[12]. This problem is overcome by the possibility of configuring Linux inserting all the modules inside the kernel and obtaining *monolithic kernel*. Given that, a group of Linux developers work on kernel while other groups write the modules allowing the computer to be connected with peripherals or to read video and audio files.

Co-ordination is thus made possible by the sensible technical choice of "keeping the kernel small and limiting to a minimum the number of interfaces and other restrictions to its future development". In this way programmers can add modules without "stepping on each other's toes".

The same Open Source project often involves individuals and organisations world-wide that contribute to make face to face communication and phone calls very rare. Asynchronous ones offered by the Internet network substitute synchronous communication media. E-mail, ftp and newsgroup have the incontestable advantage of cutting off communication costs lowering resources required for carrying out the project.

The Open Source communication media also allow for codification and storage of the produced knowledge. Hints, call and proposals for solutions, patch and comments, are tidily registered in the archives of ftp sites or mailing list. Each project benefits from a large amount of documentation produced during the development process that can be consulted by everyone interested in participating to the project, enhancing the probability of further participations.

[11] "It is possible to fit both a kernel image and full root filesystem, including all fundamental system programs, on just one 1.4 MB floppy disk"(Bovet, Cesati, 2000).

[12] The module managing the floppy disk is a typical example. It is loaded only when a floppy needs to be read and is automatically unloaded, as the floppy is not in use for a while. This leads to a great flexibility because it is possible to specify options and parameters when the modules is loaded but the speed of task execution may be reduced by the process of module loading and unloading.

Moreover, a wide series of technical support systems[13] make easier co-ordination among participants[14]. Like GNU Emacs, a development environment released by Richard Stallman under the GPL licence, such tools are almost always released under Open Source licenses and can be downloaded from the Internet for free. Given that, they are widely used in Universities and Research Centres belonging to the cultural background of the participants.

Social institutions deeply enhance the effectiveness of the above mentioned co-ordination media. First of all, software itself is a convention or a common language, in which errors are identified and corrected through the mechanism of compilation. Moreover Open Source programmers share a *common notion of validity* (Kasper, 2001), according to which the solutions that not only exhibit the best performance but also look simple, clear and logic are selected, thus guarantying non-chaotic future expansion of the work[15]. Shared behavioural rules help the perform this task. Each Open Source project includes conventions that all participants are supposed to follow. Tacit rules govern, for instance, the posting of messages on mailing lists or notification of new issues to other participants. Such rules are widely accepted in the project community and are sometimes written down and distributed to each new participant as in the case of the Debian project (Evers, 2002).

Furthermore, licences are the most important institution in the governance structure of Open Source projects. Most Open Source projects are released under *GPL copyleft licence* whose design allows to attune the incentive of the Open Source developers, serving the goals of the Open Source community better than other legal frameworks. The main characteristics of the GPL copyleft licence is its viral nature: a piece of a GPL code inserted in a given program

[13] For instance, VMS (Version Management Systems) programs trace back the story of the subsequent releases of the project and give, at each stage, a clear picture of the evolution of the project to each participant.

[15] Kasper (2001) observes that OS contributors speak about chosen solutions in terms of *beauty.* There is a sort of tacit aesthetic criterion according to which solutions have to based on an elegant logic and programming construction.

forces that program to be protected by the same license agreement; every program containing a piece of code released under the GPL, must be released under the GPL too. This forbids every piece of GPL code to be inserted in programs released under licences that do not allow the access to the source code, in general, and in proprietary software in particular. Since it guarantees the continuation of code visibility, GPL maximizes its circulation. No one can hidden the code that is the true wealth of every Open Source project. Through the code, in fact, developers signal their talent, learn new skills and exchange idea and knowledge in a common language.

In brief, hierarchical co-ordination based on the ownership of assets is not a necessary condition for carrying out complex software development tasks. On the contrary, such co-ordination would end up depressing the intellectual, aesthetic and pleasure-based motivation that seems intrinsic to the programming community.

The above mentioned framework explains the governance of activities that present some artistic or scientific intrinsic gratification. On the contrary, it is not appropriate for the so-called "non sexy" work involving, for example, the development of graphical interfaces, the compilation of technical manuals, the on line support in newsgroups and so on. According to Lakhani and von Hippel (2003), the motivations for performing *"mundane but necessary tasks"* (in the specific case giving on line help about the web server Apache) are similar to the ones governing higher level activities such as developing and debugging the software. However, it is difficult to accept the idea that these low gratification activities could be motivated by the same incentive structure than high level, creative work. The incentive and co-ordination structure of low value activities is one of the most challenging for a paradigm having creativity and reputation as fundamental values.

Although these activities display a low degree of innovativeness, they are fundamental for the adoption of Open Source software[16]. In general, while technological discontinuities are liable for the emergence of new trajectories, the spread across the system depends on a cumulative series of little incremental modifications having, in many cases, no intrinsic technological interest.

The emergence of new *hybrid* business models seems to solve this kind of problem. The new model has been spurred by the entry of companies and software houses producing free software, giving it away to customers free of charge and shifting the value from licensing agreements to additional services such as packaging, consultancy, maintenance, updating and training. This business model is widely accepted in Open Source communities and is characterised by a detailed regulation presenting innovative legal solutions, which address the relation between free and on payment services[17].

It is interesting to note that incumbents in the commercial software industry recently adopted other variants of the hybrid model. In particular software giant corporations took note of the inevitable coexistence of the two paradigms and decided to make their products compatible with Open Source platforms, to release the code of some programs and even to take part in Open Source projects. Apple has released the source code of the Mac Server OS X, Sun

[16] As an example, the two most famous Linux interfaces, KDE and Gnome, created in 1998, provided Linux users with a mouse-driven desktop environment. This kind of environment, which is standard in the commercial world, favoured the adoption of Torvald's operating system by less skilled subjects and the migration of Windows and Mac users toward it.

[17] A number of successful firms adopt variants of this hybrid business model. As an example, Red Hat, which provided NASA with software for space missions, thinks of itself as a car producer. In the automobile industry components produced by different firms are assembled in a product that satisfies consumer needs. In the same way, Red Hat puts together and compiles the source code of the Linux operating system obtaining a program ready for installation, which is distributed on a CD at less than 100 dollars together with a set of complementary applications (for instance the Gimp program for managing and changing images) and a complete documentation.

produces the Open Source suite StarOffice in order to contrast Microsoft MS Office, whereas in December 2000 IBM announced its decision to invest in Linux development for promoting the diffusion of its e-business platforms.

These hybrid business models are made possible by the proliferation of licensing agreements that follow the dictates of Open Source while not following the more extreme GPL format. In fact, provided that the fundamental Open Source conditions are respected (full access to the source code, full software usability, freedom of modification and redistribution of software), anyone can elaborate his own Open Source licence. Besides the most famous (and restrictive) GPL under which Linux and over than 80% Open Source programs are released, there are a wide set of licences. Some of them, for instance the BSD (Berkeley Software Development), allow the appropriation of software modifications. Moreover, other licences, which cannot be considered Open Source given that they protect copyright, provide the release of the source code that particular categories of users are allowed to use. This is the case of the Apple Public Source Licence by Apple and of the Sun Community Source Licence of Sun.

Besides playing an important role in the diffusion process, hybrid models solve simultaneously two problems. On one hand, they secure to potential adopters a wide range of complementary services, which are considered fundamental for reliability and usability of the software, on the other hand they provide monetary incentives for the development of "non sexy" activities, which do not fit the typical motivations of the Open Source community.

Moreover, the existence of commercial subjects using Open Source software guarantees the future survival of the software. The guarantee that software will continue to be produced and developed is a key element in the adoption decision. Users, in fact, can avoid to change applications and platform with high switching costs for infrastructures and personnel training.

The coexistence of proprietary and open software is a possible interpretative key for the success of the Open Source movement. The necessity of mixing different remuneration forms with innovative business models belongs to the intrinsic diffusion dynamic of free software.

4. The diffusion of Open Source software: negative and positive network externality effects

The recent diffusion of Open Source software has been remarkable. However, there are large differences between the server and the client market, in particular in the desktop case, in both the business and public sector markets. On the client-side Microsoft succeeded in exploiting the dominant position gained with MS Dos through the family of Windows[18] so that Open Source is not largely diffused. In the server market, particularly in web servers, the situation is quite different, with a strong or even dominant position of Open Source software[19].

This phenomenon leads us to analyse in depth the third phase of Schumpeter's taxonomy, namely diffusion. The mechanisms underlying such a rapid spread are still not very clear[20].

A more general question is what drives the pattern of diffusion of software technologies. Accepting Varian and Shapiro's definition according to which "everything that can be digitalized, that is represented by bit sequences, is information" (1999, 3), software is clearly information. A growing body of literature suggests that the diffusion of informational goods is

[18] The Windows 3.1 operating system, which inherited the characteristics of Lisa (Apple) and X Windows (Xerox Parc) in the development of mouse driven graphic interfaces, was released at the beginning of the '90s. The two graphical interfaces for Linux, Kde and Gnome were released at the end of '90s.

[19] Currently, Netcraft estimates that about 61% of Internet Web servers run Apache. The penetration rate is quite high also in the case of general-purpose servers (LAN or firm Intranets). Gartner Group (2001) assigns to Linux 10% of the servers in the United State while according to IDC (2001) Torvalds' operating system runs on 27% of them. In Europe the public sector adopts Open Source software in 8% of its servers and only 1% of its clients (Schmitz, 2001). The largest diffusion is in Germany and France where many large organisations have adopted Open Source. Focusing only on firms, Forrester (2001) finds that 56% of the 2.500 Top US Companies use some kind of Open Source software, especially on Web servers (33%). Reported figures concerning database-management programs are striking: 42% of them are Open Source while Open Source software in desktop applications is around 1%. The Forrester study shows very good prospects of development: it estimates that 20% of the present investment expenditure on proprietary licensing will be shifted on services and personnel training by 2004.

[20] Apparently, they were not clear to the founder himself: "I was aware of the transition from being the sole user of Linux to there being a hundred-odd users, but the leap from a hundred users to millions escaped me" (Torvalds, 1999).

subject to two fundamental factors: increasing returns from the production or supply side, and network externality effects, or increasing returns from the demand side.

To understand the consequences of network externality effects on the diffusion process, it is fundamental to identify their sources. Externalities can be direct, indirect or deriving from complementary services. Computer programme users are mainly affected by the first kind of externality: the classic example is the exchange of files. Anyone who has the same programme for running a given file type can read and modify files received from other people who, in turn, can do the same. Hence the importance of sharing a particular type of software – the operating system which runs the whole file system of the computer.

According to economic analysis of standards, the diffusion phenomenon is subject to path dependence and lock-in, as described in the celebrated analysis by Brian Arthur and Paul David. Generally speaking, increasing returns are defined as "the tendency for that which is ahead to get further ahead, for that which loses advantage to lose further advantage" (Arthur, 1996, 100). The presence of increasing returns can even give rise to situations where a small initial advantage is all it takes to favour a standard in such a way that it conquers the whole market.[21] Note here that increasing returns are an immediate consequence of network externalities, to the extent that Witt (1997) argues that the two can be regarded as synonyms.

The lock-in thesis has recently been the subject of heated debate. Interestingly, the nature of Open Source technologies permits the formulation of a number of points relevant to this debate.

Given the presence in the software diffusion process of the previously described direct network externalities, the lock-in mechanism seems an inevitable outcome: if a piece of software manages to gain a significant market share, a virtuous cycle is set in motion such that consumers will have even more incentive to use it, there will be an increase in the supply of complementary

[21] Paul David, for example, claims that a fundamental role was played in the establishment of the QWERTY standard for typewriter keyboards by the victory in a typing contest held in Cincinnati on 25 July 1888 by Frank McGuire, who used a QWERTY keyboard.

products (applications, maintenance) and that particular piece of software will start to dominate the market and become the standard.

Having explained the lock-in, what remains to be explained, however, is what sets it in train. Arthur (1989, 116) refers to the presence of "small events" that "by chance can give one of the two technologies the necessary advantage". If things really are like that, there is no defence against the possibility of inefficient technology establishing its dominance at the expense of another one, which, if adopted, would be superior. This is what happened with QWERTY typewriters, whose performance was inferior to the DSKs of August Dvorak (David, 1985).

The development of Open Source contradicts this prediction in that it is eroding the market share of Microsoft's dominant standard. Is this another economic absurdity? In fact the prediction of lock-in does not take into account a set of variables that have powerful effects on the pattern of technology adoption.

First of all, the mathematical demonstration of lock-in closely depends on a set of initial assumptions; changing these assumptions, Witt (1997, 753) easily manages to explain how "newly introduced technology can successfully disseminate in a market despite existing network externality".

Secondly, Arthur's hypotheses are also very restrictive and difficult to defend in the case of software. The author assumes above all that the two competing technologies arrive on the market at the same moment – the "virgin market condition"(Witt, 1997, 762)–, which is clearly unsustainable in the case of software, where the Linux operating system is competing with a decades-old predominance of systems based on and derived from MS-DOS. In the same way, it is never true that the potential market for a technology is infinite and that the decision to use it cannot be reconsidered. This applies in particular in the case of computer programmes, which have a growing but necessarily limited market (their use is not indispensable to all human activity!) and do not require extremely high fixed capital investments such that the subsequent revision of choices would involve prohibitive costs. Arthur's theory of the independence of individual choices is also hard to sustain.

Dalle and Jullien (1999) refer to local interactions to explain the dissemination of the Linux system in place of Windows NT. In their view, what is important in the choice of an operating system is not so much the total number of other individuals who use it but the number of those who do so within the group with whom the individual interacts, i.e. a local network of reference. The characteristics of such a network vary according to whether the nodes are represented by Open Source software users or by those using commercial software. A widespread phenomenon amongst the former is in fact the so-called "advocacy" theorised by leading members of the Open Source movement (Pavlicek, 1999). This is a form of one-to-one marketing whereby the users of Open Source programmes are invited to convince other members of their group of reference to do likewise and to abandon the commercial sector. Advocacy has an exponential growth: amongst its aims there is that of transforming an individual into a new disciple of the movement and hence into a potential "advocate". The role of an Open Source advocate is thus very similar to that of Witt's (1997, 763) diffusion agents who not only disseminate information about the new technology but also try to convince a group of potential users to do so simultaneously so as to contrast those diseconomies which in the presence of network externalities penalise those who first choose a new technology.

Moreover, from an economic point of view, software produced outside commercial channels is nothing more than a straightforward instance of the general issue of collective action model that applies to the provision of public good, where a public good is defined by its non-excludability and non-rivalry (von Hippel and von Krogh, 2003, 10). The new licences agreements conceived by Open Source movement have transferred such characteristics to the software. Since a piece of code is posted on the Internet, no one can be forbidden to download and use it and each new download does not affect the possibility of further downloads.

Collective action model is a useful framework for studying how diffusion of Open Source software occurs.

Fist of all, the size of the phenomenon is extremely interesting from a theoretical point of view, because it seems to contradict the classical theory, proposed by Mancur Olson, of an inverse

relation between the size of the group and the probability that collective action will be successful (Olson, 1965). The argument is that as the size of the group increases there is a progressive reduction in the probability of any defection being detected and a corresponding increase in the incentive to benefit from the good without contributing to its production. However, Olson's analysis does not take into account a set of variables whose effect cannot be ignored, above all the heterogeneity of the subjects called upon to participate in the collective action. Hardin (1982) underlines the centrality of the heterogeneity of resources and interests within the group: a small set of strongly motivated and resourceful subjects (*small group*) is often capable of overcoming the initial phases of the process of supply of the collective good, setting in motion a virtuous circle that possibly enables the phenomenon to be self-sustaining. This is the idea at the heart of the Theory of Critical Mass initially proposed by Schelling (1978), then developed in sociology by Marwell and Oliver (1993) and picked up in economics by Katz and Shapiro (1985), Economides (1996), Witt (1997) and Huberman and Loch (1999). The basic idea is that it is sufficient to have an initial group capable of producing the collective good (generally speaking, those with stronger motivation and interests so as to resist the defection of others) to discourage opportunistic behaviour in the rest of the group.

Hardin (1982) argues, in fact, that the outcome of collective action is strongly influenced by the asymmetries typical of all social processes. He contests Olson's thesis (1965) because "it is not logically possible to increase Group Size, n, ceteris paribus" (Hardin, 1982): as the size of the group increases, so too do the asymmetries within it. Of these, an important role is played by the presence of a high variability in interest (Rossi, 1998, 79). If "the interest is very concentrated and the subjects are highly interested, they can constitute an efficacious subgroup capable of supplying the good without the co-operation of others, even though they form only a small fraction of the collective" (Hardin, 1982, 79). The efficacious subgroup will thus procure the good that can then be consumed by all members of the group. In the case of Open Source software, the efficacious subgroup is made up of hackers. Their major resource is their indisputable know-how in computer science, their most marked interest is code writing as an

intellectual activity the possibility to produce by themselves those goods they cannot find on the market.

Their supply action is also made very effective due to the fact that the source code good is non-rival in terms of consumption. The benefits it brings are invariant in relation to the number of users: anyone can download code from the Web, compile and use it without reducing other people's capacity to do so. This further confutes Olson's thesis. As Chamberlain (1974) demonstrated in the case of non-rival collective goods, the quantity supplied increases with the size of the group.

Hardin's conclusions provide an account of how Open Source software is produced by the most interested hackers, but it does not explain why other individuals also collaborate in its production in an increasingly extensive process of co-operation. For this it is necessary to refer to Marwell and Oliver's work on the theory of Critical Mass.

Here the role of strongly interested subjects is not to provide the good entirely themselves, but to create the necessary conditions for production to become easier. Their task is to "be the first to cooperate", enabling the difficult initial start-up phase of collective action to be overcome. This is the role played by those who originate an Open Source project, deciding which problem to work on and implementing the central nucleus of the code, which will then be subsequently modified, perfected and enriched by other contributions.

The diffusion process of Open Source software seems, then, to satisfy the hypotheses at the heart of the existence of Critical Masses in the spread of new technology, which permit an alternation of standards as opposed to the long-term dominance of inferior technology predicted by Arthur. According to Schelling (1978, 91), the term is borrowed from physics and indicates the quantity of radioactive combustible required to set off nuclear fission. Transposing this concept to the economic and social sphere, one talks of the Critical Mass effect if, when certain variables characterising a process rise above a given threshold, the phenomenon explodes, so that the system moves from the stable equilibrium in which it was initially positioned and

assumes another stable equilibrium. In technology diffusion models, the variable is represented by the number of people who adopt the new technology.

Dalle and Jullien (1999) have carried out computer simulations to analyse the Critical Mass phenomenon – which they call *threshold effects*– in the diffusion path for Linux. They observe that its emergence depends on the value assumed by a particular parameter α, which is a measure of the "preference of users for standardisation" within their network of reference: the greater α is, the more potential users are inclined to adopt the current standard, that is Windows NT. This parameter is therefore also a measure of the proselytising power of Open Source users: the lower α is, the greater is their strength within each local network. The simulations reveal the presence of threshold effects when the value of α is sufficiently low.

Other authors agree in recognising the importance in this field of expectations regarding the performance of technology and the final size of the network of users (Katz and Shapiro, 1985, 426; Bensen, Farrell, 1994, 118). Analysing the diffusion of two competing technologies whose performance grows as the number of people adopting it grows, Huberman (1999) concludes that the waiting period before the achievement of Critical Mass decreases in proportion to the degree of optimism of people's expectations. The transition from the initial equilibrium where everyone uses the old technology to the one where the new technology has replaced the old one also takes place more quickly the more heterogeneous the subjects are. Huberman's simulations demonstrate that if there is a small group of individuals with double the propensity for innovation in relation to the group average, the Critical Mass is reached almost immediately. This once again underlines the role of hackers and their culture in the Open Source movement. Their existence is important not only to explain how the phenomenon arises and finds an economic application, but also to explain how it spreads.

Once again the development of the Open Source phenomenon makes it possible to clearly see the importance of heterogeneity in diffusion processes. The presence of active minorities does not have a great impact if one assumes models with homogeneity amongst agents (because the

effect is mitigated by communication, which occurs randomly between all members of the population) or if one assumes increasing returns to adoption with high costs for revising choices. If on the other hand one assumes agent heterogeneity, local interaction and the revision of technological choices, it is possible to find phenomena involving the overturning of established standards.

Another effect must be considered in the diffusion of Open Source. While network externality effects clearly favour incumbent software producers due to the size of installed base and compatibility problems, a *positive* network externality effect is also in place. This is due to the possibility for adopters to benefit from the legitimate access to a large pool of usable software tools, whose value is proportional to the number of adopters of Open Source software. We therefore posit two opposing network externality effects: a negative effect inhibiting adoption and a positive effect favouring adoption. We will consider the difference between these two effects by modelling the negative effect as a continuous variable proportional to the number of adopters of Open Source (then a negative effect at the beginning of diffusion, given the prevalence of proprietary software) and the positive effect as a threshold variable.

Finally, we should not ignore the competitive reaction from incumbent software producers. While Open Source software is promoted by a large voluntary community, commercial software producers may try to defend their installed base by investing additional resources in R&D and improving the quality of proprietary software.

5. Open Source diffusion: A simulation model

From the discussion above we conclude that the adoption of Open Source and its ultimate diffusion are influenced by its perceived intrinsic value, the negative network externality effect coming from the dominant standard, the positive network externality effect coming from the access to the community of programmers, and the competitive reaction of incumbents in the commercial software industry.

This section presents the results of an agent based simulation exercise, modelling the adoption decision of a population of heterogeneous interacting agents. The model is implemented using the SWARM platform [22].

Starting from a situation in which the proprietary software has a prevalence of 100%, each of the N agents (N = 1,000) must decide whether to adopt Open Source software, giving up the dominant standard, or to continue to use the proprietary software. Agents are also allowed to use both technologies. Starting from the dominance of proprietary software helps to make clear the conditions under which Open Source may diffuse. Each agent makes a decision on the basis of an explicit adoption function formulated as follows:

$$F = \alpha \cdot (1-\rho) \cdot IV + \left[(1-\alpha) \cdot NE + \Delta \cdot CO\right] \quad with \quad \alpha, \Delta \in [0,1] \quad (1)$$

The function F is normalised in the interval [0,1] and is composed by two parts. The former part (α·IV) deals with agents' evaluation of the intrinsic value of Open Source software, while the latter ($\left[(1-\alpha) \cdot NE + \Delta \cdot CO\right]$) deals with the behaviour of other agents in the system.

IV (intrinsic value) is modelled as a random continuous variable in the interval 1-5, representing in increasing order the value that each agent attributes to the new technology with respect to the old one. We take into account several distributions of the variable, reflecting different attitudes towards the Open Source technology. First of all, following the classical literature on the diffusion of innovations (Rogers, 1995) and the *probit model* approach (David, 1969; Geroski, 2000), a normal distribution (NORM) is assumed. Then we examine the case of biased beliefs, towards the old technology (exponential distribution with mean equals to 1: EXP_1) and towards the new technology (exponential distribution with mean equals to 4: EXP_4). The latter

[22] Swarm is a collection of software libraries, which provide support for simulation programming. The Swarm project was started in 1994 by Chris Langton, then at Santa Fe Institute (SFI)in New Mexico. The aim was to develop both a vocabulary and a set of standard computer tools for the development of multi-agent simulation models (so-called ABMs, short for Agent-Based Models). See http://www.swarm.org for additional information.

distribution captures the presence in the population of *Open Source advocates* characterised by a strong appreciation for the new technology that they try to transmit to other individuals through social contacts. Finally, the hypothesis of a uniform distribution (UNIF_1_5) in the interval is considered.

Parameter ρ captures incumbents' reaction. Each time the diffusion of free software increases by 1%, the incumbents supplying proprietary software react by investing in R&D and improving the quality of their products. In relative terms the agents' evaluation of Open Source software is reduced by a given percentage (ρ). Briefly, we model competitive reaction as a modification in the intrinsic value that agents attribute to the new technology.

Variable NE addresses the network externality effects in the use of computer software. In order to take into account the decreasing amount of utility brought by the marginal agent, NE increases at a decreasing rate:

$$NE = \sqrt{\frac{n_a(t)}{N}} \quad (2)$$

Where $n_a(t)$ is the number of Open Source adopters at time t.

Variable CO accounts for the co-ordination effect or positive network externality effect. When the number of adopters is sufficiently large, not only they can exchange files, but also can work together in writing complete computer programs. Co-ordination effect is triggered when adopters reach a given percentage of the total (γ):

$$CO = \begin{cases} = 0 & if \quad n_a(t) < \gamma \\ = \dfrac{n_a(t)}{N} & if \quad n_a(t) > \gamma \end{cases} \quad (3)$$

Agents are structurally and socially heterogeneous, in the sense that are endowed with social contacts that allow the updating of the evaluation of Open Source technology. The updating mechanism is quite simple. When an agent has a contact (according to the simulation structure),

the mean between the value of her state variable and the state variable of the contacted agent is computed. This new value is assigned to the state variable of the contacting agent.

At time t = 0 contacts start. After having a contact, agent *i* evaluates her own adoption function:

- If $F < 0.33$ *i* continues to use only the proprietary software
- If $F \in [0.33,\ 0.66]$ *i* selects both technologies
- If $F > 0.66$ *i* adopts Open Source giving up proprietary software[23].

In sum, we build a model in which the adoption decision is made explicit. A new technology is adopted if its intrinsic value is high, if other members of the population adopt and if there are direct benefits from co-ordination arising from exchanging pieces of code. The relative importance of these two effects is managed by parameters α and Δ. We are interested in understanding under which conditions, as reflected in parameters, Open Source will spread in the population of potential adopters. In this version of the model we do not allow for choice revision, which might be considered in the development of the research.

Note that the setting does *not* mimic a classical epidemic diffusion process. Once adopted, agents do not infect others with given probability. Rather, they average their own evaluation with that of the contacted agent, so that it is still possible for them to *decrease* their evaluation, so becoming *less* infectious, even though they continue to use the new technology. Therefore we do not expect to find a classical S-shaped dynamics. Adoption is not triggered by contact alone. As a result, we are more interested in studying the prevalence level (i.e. the penetration rate of Open Source in the limit) than in plotting the cumulative diffusion curve over time.

In the first group of simulations (benchmark case) parameters take the following values:

$$\alpha = 0.75,\ \gamma = 0.02,\ \Delta = 0.2,\ \rho = 0.10$$

This case represent a situation of moderate negative and positive network externality, and strong competitive reaction. Table 1 reports simulation results, giving the total number of adopters out

of 1,000 agents in 7,300 runs[24]. In order to give robust results, we explore all initial distributions.

	Proprietary software	*Both technologies*	*Open Source Software*
Exp_1	994	6	0
Exp_4	0	626	374
Norm	0	865	135
Unif_1_5	0	867	133

Table 1 Benchmark case

Several points should be noticed. Open Source diffusion indeed depends on the *initial* distribution of intrinsic values assigned to the technology by agents. In general, a moderate negative network externality effect and a strong competitive reaction are not sufficient to block the diffusion of Open Source, even starting from a normal or uniform distribution.

However, a biased belief distribution against Open Source (Exp_1) may in fact block the entire diffusion process. Since a threshold for triggering positive network externality effect is required ($\gamma = 0.02$), the proportion of agents that start adopting on the basis of a favourable perception of Open Source is not sufficient to counterbalance the competitive reaction by incumbents.

[23] The definition of boundaries is somewhat arbitrary, but it reflects equal weight assigned to the different alternatives of adoption. Technically speaking it is possible to adopt both commercial and Open Source software on the same hardware, given the wide availability of compatible programmes.

[24] There is not an immediate translation of the model clock into chronological time. Since each run represents a set of social interactions, we may assume that they take place several times per day. In this case the simulation amounts to a few years. Other interpretations are also legitimate.

Data refer to one realisation of the stochastic process for each case. In most cases we controlled the mean for several realisations (i.e. 10).

The presence of a biased belief distribution (Exp_4) does not influence the total number of adopters but only its internal distribution. Finally, under all distributions, with the exception of Exp_1, the dominant outcome is one of coexistence between the two technologies.

Given this benchmark, the parameters are altered one by one and set to one or zero in order to evaluate their influence on the diffusion, keeping other parameters constant. We study only the most favourable distribution to Open Source, Exp_4. By doing so we focus on the conditions under which Open Source may diffuse starting from a favourable initial distribution of beliefs, giving an *upper* bound to the diffusion process. Results of the sensitivity analysis are listed in Table 2.

	Proprietary software	*Both technologies*	*Open Source Software*
$\alpha = 1$	382	392	226
$\Delta = 0$	349	522	129
$\gamma = 0$	0	626	374
$\rho = 0$	0	545	455

Table 2 Variations in parameters

When negative network externality are not present ($\alpha = 1$) adoption is influenced only by agents' beliefs, co-ordination effects and competitive reaction. In this setting, even though initial beliefs strongly favour Open Source, incumbents still manage to retain 38% market share in the limit. At the same time, when the co-ordination effect is set to zero, a similar result is found. Clearly setting to zero the parameters α and Δ eliminates their qualitative difference, i.e. the former representing in a continuous way the negative impact of installed base of commercial software and the latter representing in a discrete way the positive effect of Open Source

communities. Keeping in mind that the initial distribution is strongly biased in favour of Open Source, the resilience of commercial software is remarkable.

On the other hand, when there is not a threshold for the positive co-ordination effect ($\gamma = 0$) and there is not competitive reaction by incumbents ($\rho = 0$), Open Source saturates the market, although commercial software survives in more than 50% of cases as a second source.

The implication from Table 2 are clear: given a distribution of beliefs biased towards Open Source and in the absence of incumbent advantages the only way for commercial software to control the market is to engage in a fierce quality competition supported by massive R&D efforts. In the absence of competitive reaction Open Source ends up in dominating the market.

The results of the simulation exercise confirm that the diffusion of a network technology in presence of a well-established standard is difficult and takes a long time. Starting from this consideration, the simulations suggest that the diffusion of Open Source on one side depends on the initial distribution of the beliefs, while on the other side relies on the activation of several sources of network externality. Incumbents are in the position to slow or even block the diffusion of the competing technology, by investing aggressively in R&D and/or leveraging on the large installed base. A robust general result is that under many plausible conditions commercial software and Open Source software are likely to *coexist* even in the limit.

6. Conclusions

This work shows that the peculiarity of the Open Source movement can be explained using recent developments in the theories of collective action, of co-ordination in the absence of a central authority and of the diffusion of technologies in the presence of network externality. This has made it possible to throw light not only on how the phenomenon arises and finds a vast economic application, but also on the mechanisms that underlie its massive diffusion. Many questions do, however, remain open. There clearly needs to be more in-depth analysis of co-ordination mechanisms, with particular reference to the precise functioning of Open Source

projects, in order to understand the role played in them by the project leaders and by the minor figures who do the non-sexy work**.** The search for an explanation of why this work is done is one of the most interesting challenges that remains for a full economic understanding of the Open Source movement. While both sociology and the economic theory of incentives are able to provide an account of how a Finnish university researcher managed to write the Linux kernel without being paid, it is more difficult to understand what motivates a programmer to work on the realisation of a boring graphic interface to make Linux more user-friendly. This is a fundamental point for two reasons: on the one hand, the tendency of Open Source programmes to become more user-friendly will enable their diffusion in increasingly broad bands of the population, on the other, user-friendly programmes and user assistance are the core business of many of the new companies who, by managing to make profits through Open Source, have demonstrated that in the case of software the word "free" has the meaning of "free speech" and not "free beer". The mix of commercial and free solutions that goes on through the birth of new companies and the richness of licensing arrangements are other interesting phenomena to be examined in the near future.

One of the main implications of our analysis and of the simulation exercise is that under many plausible situations commercial software and Open Source are likely to coexist in the future.

The Open Source phenomenon is certainly not the only new information society to throw up interesting economic problems. The transformation of software from private good to collective good realised by Torvalds and the ensuing movement seems to have made Kenneth Arrow's frequent remark come true, when he warned that in contexts where there is knowledge-intensive innovation, the goods no longer have a fixed nature but are to different degrees *both* private and appropriable, and public and universal. The fact that economic theory is best equipped to work on simple cases – with highly specialised theories for private goods (markets) and public goods (public finance) – and struggles with complex cases is obviously a problem. Fortunately, it is not a problem of the real world, which quickly moves on even without economic theory.

References

Arthur, W.B., 1989. Competing technologies, increasing returns, and lock-in by historical events. Economic Journal 99, 116-131.

Arthur, W.B., 1994. Increasing returns and path dependence in the economy, University of Michigan Press, Ann Arbor.

Arthur, W.B., Ermeliov, Y., Kaniovski, Y., 1983. On generalised urn schemes of polya kind. Cybernetics 19, 61-71.

Baldwin, C.Y., Clark, K.B., 1997. Managing in the Age of Modularity. Harvard Business Review 75(5), 84-93.

Bensen, S., Farrell, J., 1994. Choosing how to compete: strategies and tactics in standardisation. Journal of Economics Perspectives 8(2), 117-131.

Bonaccorsi, A., Rossetto, S., 1999. Software reusability: a real option analysis. International Journal of Software Technology Management, fall.

Brooks, F. P., 1975. The mythical man- month: essays on software engineering, Addison- Wesley, New York.

Chamberlain, J., 1974. Provision of collective goods as a function of group size. American Political Science Review 68, 707-716.

Dalle, J., Jullien, N., 1999. NT vs. Linux or some explanations into economics of Free Software. Paper presented at "Applied Evolutionary Economics", Grenoble, June 7-9.

David, P.A., 1969. A contribution to the theory of diffusion. Centre for Research in Economic Growth Research Memorandum 71 (Stanford University, Stanford).

David, P.A., 1985. CLIO and the economics QWERTY. American Economic Review 75, 332-337.

Economides, N., 1996. The economics of networks. International Journal of Industrial Organization 16(4), 673-699.

Edwards, K., 2001. Epistemic Communities, situated learning and Open Source software development. Paper presented at the workshop "Epistemic Cultures and the Practise of Interdisciplinarity, Trondheim, June 11 – 12.

Evers, S., 2000. An introduction to Open Source Software Development, Technische Universität Berlin, Fachbereich Informatik, Fachgebiet Formale Modelle, Logik und Programmierung (FLP).

Franck, E., Jungwirth, C., 2001. Reconciling investors and donators – The governance structure of Open Source. University of Zurich Working Paper (University of Zurich, Zurich).

Gartner Group, 2001. An end-user analysis. Linux server market share: where will be in 2001 and how will grow?

Geroski, P.A., 2000. Models of Technology Diffusion. Research Policy 29 (4-5), 603-625.

Ghosh, R., Prakash, V.V., 2001. The Orbiten Free Software Survey. First Monday.

Glass, R., 1999. Of Open Source, Linux and Hype. IEEE Software 16(1), 126-128.

Green, L., 1999. Economics of Open Source software. http://www.badtux.org/eric/editorial/economics.html.

Hardin, R., 1982. Collective action, John Hopkins University Press, Baltimore.

Hecker, F., 1999. Setting up shop: the Business of Open-Source software. IEEE Software 16(1), 45-51.

Hurberman, B., Loch, C., 1999. A punctuated equilibrium model of technology diffusion. Management Science 45, 160-177.

IDC Report, 2001, available at http://dailynew.yahoo.com/h/zd/20010611/new_report_questions_linux_server_claims_1.html.

Katz, M.L., Shapiro, C., 1985. Network externalities, competition, and compatibility. American Economic Review 75, 424-444.

Koch, S., Schneider, G., 2002. Effort, co-operation and co-ordination in an Open Source software project: GNOME. Information Systems Journal 12(1), 27-42.

Lakhani, K., von Hippel, E., 2003. How Open Source works: "Free" user-to-user assistance. Research Policy, Forthcoming.

Lerner, J, Tirole, J., 1999. The simple economy of Open Source, NBER Working Series, Working Paper 7600 (NBER, Cambridge, MA).

Marwell, G., Oliver, P., Prahal, R., 1993. The critical mass in collective action. A micro-social theory, Cambridge University Press, Cambridge.

Olson, M., 1965. The logic of collective action, Cambridge University Press, Cambridge MA.

Pavlicek, R., 1999. Keys to effective Linux advocacy within your organisation. http://users.erols.com/plavlicek/oreilly/als-fullpaper-1999.txt.

Perkins, G., 1999. Culture clash and the road of word dominance. IEEE Software 16(1), 80-84.

Raymond, E., 1999a. The cathedral and the bazaar. http://www.redhat.com/redhat/cathedral-bazar/.

Raymond, E., 1999b. Breve storia degli hacker. In Di Bona, S. Ockman and M. Stone (Editors), Open Sources: Voci dalla Rivoluzione Open Source (Apogeo, Milano).

Rogers, E., 1995. The diffusion of innovations, 4th Edition, Free Press, New York.

Rossi, C., 1998. Il Problema dell'Azione Collettiva nella Teoria Economica: una Rassegna di Alcuni Recenti Contributi. Degree Thesis. Economics School, University of Pisa, Italy.

Sanders, J., 1998. Linux, Open Source and software's future. IEEE software 15(5), 88-91.

Schelling, T. 1978. Micromotives and macro beavhior, Norton, New York.

Schimitz, P.E. (2001) Study into the use of Open Source software in the public sector. An IDA Study (Interchange of Data between Administrations).

Schumpeter, J., 1934. The theory of economic development, Harvard University Press, Cambridge.

Torvalds, L., 1999. Il vantaggio di Linux. In Di Bona, S. Ockman and M. Stone (Editors), Open Sources: Voci dalla Rivoluzione Open Source, Apogeo, Milano.

Ullman, E., 1998. The dumbing down of programming. Paper presented at 21st Salon, 13 May, http://www.salonmagazine.com.

Varian, H., Shapiro C., 1999. Information Rules: A Strategic Guide to the Network Economy, Harvard Business School Press, Cambridge.

Von Hippel, E., von Krogh, 2003. The private-collective innovation model in Open Source software development: issues for organization science. Organization Science, Forthcoming.

Wallich, P., 1999. Cyber view the best things in the cyberspace are free. Scientific American, March, 44.

Witt, U., 1997. Lock-in vs. critical masses. Industrial changes under network externalities. International Journal of Industrial Organisation 15, 753-772.