Open Source Software: A New Approach to Software Development

Michele Biavati, Instituto Tecnologico Autonomo de Mexico

Abstract

The productions of opens source software can be considered an example in which not only monetary and selfish motivations lead people's behavior. Beside the possibility of signaling one's own abilities offered by this new production system, altruism and satisfaction derived by belonging in the Open Source Community constitute a fundamental aspect in programmer's motivations.

The particular motivations of agents and the rewards mechanism (both intrinsic and monetary) are such that programmers are very efficiently organized in a multidimensional network that includes not only programmers, but also user and tester, and often these roles interchanges. At a close view seems also that this organization is also able to produce software that seems of better quality that the proprietary one.

The presence in the scenario of commercial companies that make profit with Linux plus the networks externalities that Open Source Software create are other two important keys to understand the success and the diffusion of the Linux operating system.

Moreover, organisational arrangements constitute an essential part of the development models followed inside technological paradigms. I specifically analyzed the different development's management systems that characterize Microsoft and Linux production's organization. Hierarchy and decentralisation constitute different, although in most occasions complementary, kinds of organisational settings. Hierarchical organisations are based on a top-down approach to management: information (in the form of "commands") flows down the hierarchical chain to those subordinated, assigning them to different tasks. In the case of decentralisation, semi-independent units work in a more autonomous way (albeit subject to a certain hierarchical control in most occasions, that tries to co-ordinate the activities of the independent productive "cells").

Through decentralisation, it is easier to determine the output for each of the working units. This makes management easier (in a sense, the advantages of decentralisation are similar to those of modularity: it becomes easier to determine what is going wrong and where) and facilitate the linking of rewards to performance, providing incentives for workers to increase their effort, and therefore helping to solve the typical problems of principal-agent found in most firms.

Through decentralisation "informational responsibilities" are more efficiently distributed across the firm. The manager is, to a certain extent, relieved of the burden of coordinating the productive process to its finest degree (for what an enormous amount of information is required for this). Workers have a greater opportunity to take part in the decision-making process in the areas in which they are most knowledgeable.

In the same way, process innovation is favoured: there is no need for redesigning enormous and very complex production processes in order to introduce innovations. Small changes in the operating units can increase their performance without affecting the rest of the productive system. Workers, who are better informed about what they are doing, can introduce small-scale innovations. The human capital of the whole organisation is exploited in a more efficient and comprehensive way.

Communication and control of activities are deeply linked to the hierarchical or decentralised nature of an organisation. Management consists of a transmission of information that tries to influence an actor so that he/she performs a task in a way that is considered to be more efficient, given the objectives of the organisation. If we consider this definition of management, we could say that in Linux there is no management at all. In the case of Linux the only flow of information that goes top-down is the release of new versions as a set of “information”. The feedback that the centre obtains makes up the actual components that will be integrated into the next release. Moreover, the programmers involved in Open Source explicitly use comparisons about the evolution of alternatives in terms of selecting the best and weeding out of the worst.

In the case of more formal management, there are decisions at the top which are transmitted to the bottom and then feed back from the bottom to the top, reporting on the results of the activities previously “ordered”. The need for information processing will be stronger in this kind of arrangement: the manager needs to determine what is needed and then ask his/her subordinates to do it. Then the manager obtains information about the results. In contrast, in the first case Open Source, the “manager” just tells the community what the situation is and the community gives him/her the potential answers from which they should choose.

The structure that we could more clearly associate to Linux has, therefore, some important advantages in the dynamic and complex world of software development. First, it becomes easier to avoid organisational inertia and adapt more flexibly to changes in the environment. Decisions come from the bottom–where more knowledge will be accumulated–that present a more accurate perception (and conscience of the importance) of potential future alterations in the competitive, technical and social setting.

Another fundamental aspect analyzed is the hybrid market structure of the Open Source products. Although the basic software is freely available to anyone, there exist companies and software houses that base their activities on the commercialization of Open Source software; more specifically, they sell a “package” that, in addition to the free software, includes consultancy, manuals, maintenance, updating and training. This business model is widely accepted in open source communities and it is characterized by a detailed regulation presenting innovative legal solutions which address the relation between free and on-payment services.

Besides playing an important role in the diffusion process, those companies simultaneously solve two problems: first they secure for potential adopters a wide range of complementary services, which are considered fundamental for reliability and usability of the software. Second, they provide monetary incentives for the development of “non-interesting” activities, which do not fit the typical motivations of the open source community. Moreover, distribution through the market can thus reduce other (later or perceived) costs, even if it is cost to obtain the software package. In addition, it gives legitimacy as a “real” alternative

INTRODUCTION

In recent years a new means of software development has appeared in the computer industry. This new approach, known as open source movement, is based on a crucial right for all to read and access the software’s source code (freely available on the internet) and the opportunity to modify and develop the programs and then redistribute new versions. The idea behind this approach can be described in this way: “when programmers can read, redistribute and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.”[1]

Open source movement can be considered as a new innovation that is in opposition to the traditional property rights approach. It can be defined as a culture of individuals devoted to the free development and distribution of software. These individuals believe in mutual help and the advantages of sharing software code.

Open source software code constitutes a public good, and, on first inspection, its production seems paradoxical: it is a public good provided by volunteers that seems to be of better quality than commercial software. Moreover, it is free in two aspects: it has no direct cost to the user and gives the user the freedom to modify the program.

It is clear that once produced, those who have not contributed to their production can freely use these goods. In particular, open source code is a non-rival good in terms of consumption. Because it seems profitable to utilize the contributions of others, obtaining benefits without fronting any costs, the free rider problem should arise. But, if everyone were to follow this argument, the software would not be produced. Analyzing the motivation that drives people to write and provide free source code will explain the reason why this does not happen in practice.

In practice what is happening is that the presence of a few highly motivated subjects allows the initial phase of supply the collective good to pass, where cooperation overcomes benefits. After this phase, more and more agents realize that contribution is profitable for more and more agents. All this sets in motion a virtuous circle that enables the phenomenon to be self-sustained, converging towards a new equilibrium in which all members choose to cooperate.
Here the role of strongly interested programmers is not to provide the good entirely by themselves, but to create the necessary conditions for production to become easier. Their task is “to be the first to cooperate”, enabling the difficult start-up phase of collective action to be overcome. This framework very well explains the dynamics of open source projects. They originate from a small group of subjects working for achieving a deliberate goal of giving. If the group succeeds in finding

[1] This idea can be found on the web: http://www.opensource.org

good solutions, the results are posted on the Internet and the project is advertised on mailing lists and newsgroups. Contributors, in turn, decide which problem to work on.

Open source communities are organized in a collaborative network that communicates through the internet, avoiding any kind of barrier to the free flow of information between developers and users. This may be the first example wherein the internet enables cooperation on a scale that changes market dynamics.

Open source software is a certification mark. Software, in order to be denominate open source, has to respect the following rules[2]:

- Free redistribution, without restriction or fees.
- Program must include and allow distribution in source code as well as compiled form.
- Modified and derived works have the same right of distribution.
- Integrity of the author's source code, enabling users to distinguish between base source (e.g. non modified) and patch files.
- No discrimination among persons or groups.
- Distribution of license follows with all redistributions.
- License must not be specific to a product (e.g. dependent or being part of a software distribution (package)).
- License must not contaminate other software (e.g. must not demand that all other software is also open source).

In other words, these rules give users the freedom to download the software and the source code from the internet, to modify the program to suit their needs and to redistribute modified versions[3]. More clearly, there are two senses in which Free Software is "free": it has zero direct cost to the user and it provides the freedom to modify the software.

LINUX

Linux is an operating system based on UNIX, the main programming language used in the academic world. An operating system is the main interface that allows a user to communicate with the computer and allows the computer to communicate with other computers that are connected through a network.

Linux, one of the most recognizable of the open source projects, started when Linus Torvalds, motivated by both his frustration with existing software and a desire to write software programs, published his operating system and the source code on the internet, involving other people in his

[2] http://www.opensource.org

[3] Among the aims of this paper is not included a deep analysis of the scope of open source licensing. For an exhaustive review of different types of licence agreements and the related literature refer to Lerner and Tirole (2002b).

project. He asked hackers to provide comments, suggestions and new ideas in regards to his program.

The Linux project, since first introduced, has received a vast audience of people. It works through a decentralized and scattered network of programmers, the “community of developers”. This group of volunteers contributes to the development of Linux.

Torvalds, as the leader of the Linux project, releases different versions of the operating system. These are freely available on the Internet. Those who are interested, having access to the source code, can attempt to improve upon the program, test it and add new features. Due to the fact that different users have different needs, they write programs to better suit their specific needs; each developer determines the specifications and design of new features and functions. What made it easy for programmers contribute to the Linux project is a fundamental characteristic of the software: Linux is highly modular; it consists of separate programs, each of which is independent to the others. One of these programs is the kernel - the program that directly addresses and controls the hardware. The other modules can “communicate” with the computer through the kernel. The fact that Linux has a modular structure is very important. This enables users to work on a small part of the program and to submit improvements while not necessarily knowing the entire source code. Moreover, the community can decide which parts to develop.

Any kind of improvement, correction or feedback regarding errors and bugs is submitted back to the leader, who then decides which contributions to incorporate into the new version of the program. The actions of the programmers are organized by the contributor’s mailing list - the primary place for technical discussions pertaining to aspects of the development. Torvalds then makes final decisions about which developments to include in the new version. Each version contains a “credits file” in which the name of each contributors and which part was developed is recorded.

Torvalds acts as “benevolent dictator” and his decisions are influenced by the opinions, suggestions and criticisms of the community. It is important to emphasize that this group is composed of highly competent individuals; these programmers participate because they have the interest and the ability to do so. In making these choices, Torvalds influences the development of the project by choosing a particular solution and asking for efforts in particular fields. He also has to be willing to listen to suggestions for improvements as well as criticism. He plays the role of a project leader of a global group, but one where members only participate if, and when, they have the interest and the ability to do so. He can lead and choose but not command. Moreover, the ideas in an Open Source project are exchanged democratically with respect to free communications but decisions, as we have seen, are not completely democratic. However, the non-democratic decision process is accepted due to the fact that the results are open for anyone to change.

The fact that the source code of the software is open permits the users to customize the program in order to more efficiently satisfy their needs. This possibility not only gives much flexibility to users, but it is essential to the development of the software: individual modifications that are submitted back to the community allow open source software to evolve into a high quality product and to expand its features.

Linux is released at a very fast speed and reliability is achieved by external testing by the users. Each version is associated with a different degree of stability (it depends how completely the version has been tested and debugged). Users, depending on their interest and objectives, can choose which versions they prefer to use. This release policy permits the rapid development and production of high quality (in terms of stability) software.

What makes the Linux project a successful way to produce an operating system are some internal characteristics on which Linux is based: among programmers there is a general idea that authority comes only from competence (not from any other factors). Many of those involved in open source software believe (Raymond 1999) that the way in which Linux is developed is a very efficient methodology to produce software. Everyone can join the creative and democratic community. They can contribute, test, give suggestions and provide feedbacks.

Moreover, the fact that the solutions are chosen through the consensus of highly specialized programmers means that the best solutions are accepted for the source code. In order that he/she not lose a consensus, the leader has to accept the decisions that are considered to be the most right by the community. In addition, the fact that every user is a potential contributor (even if it means alerting the community to an error) ensures that the program is tested by thousands of users. This is a very efficient way to develop software, with extremely stable results. A large audience tests the programs in real scenarios, correcting errors during their use and then provides immediate feedback. This process also works for the production of entirely new modules.

New contributions are published on the web on a regular basis. They are analyzed by the community before they become part of the newly released software. As Raymond's (1999) claims, when programmers are allowed to work freely on the source code of a program, exchanging files and ideas through the internet, the software will be improved because collaboration helps to correct errors and enables the software to adapt to different needs and hardware platforms[4].

A precondition of the success of the open source projects is that it is based on the Internet. This is because the internet provides, at very low cost, simple but reliable communication tools that are available worldwide. This permits real time communication and helps to cut the cost of sharing information. In fact, the infrastructure investment for taking part in the movement is almost zero. Most of the programmers have their own computer for studying or working – some will even write the code at work.

MOTIVATIONS

One of the first questions that arise when analyzing the open source movement is why thousands of people volunteer their services for the production of a public good. There is a wide range of motivations to contribute, that range from altruism and personal benefit to idealism and monetary pay offs.

[4] http://www.opensource.org

The open source community mainly consists of hackers that began their involvement in Linux because they thought it was a "cool" project. The fact that Linus Torvalds opened the development of his software to many people was a big psychological incentive for hackers to participate.

The development of a network of users/developers corresponds with the ideology behind the hacker culture and computer science students – universal freedom and access to information. It concerns the rights and responsibilities of individuals with the idea that free information should be accessible to all; it is related to "the recognition of democracy as the fundamental of modern society, better and more liberated means of communication, and an accelerating information flow creating a higher degree of complexity in society"[5]
The early hacker culture has been in favor of non-commercial, non-proprietary software, where recognition of technical expertise, as well as simply discovering if something will work or not – rather than direct financial returns - have been important incentives for writing code.

Linux gives the hackers the challenge of improving software for their needs and for the benefit of the community. The good of the community enters into the preferences of the individual contributors. For this fact, the open source community is often described as a gift-culture (Mauss 1959). The publication of new parts of the program or the contribution to the development of the software is perceived as a gift, characterized by the psychological benefit that the donator obtains in helping the other members and giving his contributions for the benefit of the community (Raymond 1999). Contributing to the production of the software allows users to make and preserve social links and implies the duty of reciprocation. In the developers' community, this happens even if the gift is not direct to an individual person but to members of the community. People believe that contributions help to form a tacit agreement within the community to contribute in the future.

One of the main reasons for developer's to participate in the open source project is to create a particular software, one that does not exist in the markets, to better satisfy their own needs. Programmers want to be able to strip the source code down to exactly that set of functions that they need.

Moreover, for a hacker programming is an entertaining task - one from which he/she obtains great joy and can display his/her own abilities. To a programmer, code writing is an art from which they obtain artistic gratification. As Ulman (1997) shows, programmers experience a strong personal satisfaction in creating "something that works". In addition, in a recent study of people involved in open source software, results show that more than 70% of open source's contributors lose track of time while programming. In these cases, "writing open source code is not a cost but a benefit, not investment but consumption"[6]

From certain points of view, open source projects can be analyzed as a social movement. Social movements can be defined as "efforts by a large number of people to solve collectively a problem that they have in common" (Toch 1965). This definition fits with the aims of the open source movements, such as the desire to modify the programs to satisfy personal needs, improve the quality of the software, protecting the diversity of software solutions against the strong

[5] Pedersen 2001
[6] Osterloh et al. 2002

dominance of large economic enterprises, specifically competing against Microsoft's supremacy (Microsoft has been a common enemy since it represent ideals that are not relfected in the Open Source movement). Clearly, the more important these aims are for the programmers, the more the programmers are willing to contribute.

Open source developers are highly suspicious about the "customerization of the computers" caused by Microsoft. Moreover, programmers often believe that their participation is essential for the success of the project. Along with this instrumental component, we can consider the perception of the programmers to have the right abilities for a certain task: they believe that their contributions will be highly useful to the community. This is due to the fact that they are among the few people capable of producing the best results. In other words, they attribute a great importance to their participation in order to achieve the project's goals.

A further aspect connected with the motivations to contribute in the open source projects, is similar to the motivations that leads the scientific community to share their results. Disclosing and sharing results in the open source community allows programmers to receive feedback from their peers. This in turn permits programmers to improve their results and correct their errors. Considerations and comments emerged in the discussion groups stimulate more developments and are considered a challenging way in which one can access a wide audience composed of the best programmers. This provides critiques and stimulation and gives the programmers the possibility to learn from the suggestions of others. In addition, due to the system of credits that give a clear visibility to contributors, disclosing results on the internet enables programmers to gain intrinsic reputation and recognition, signaling their abilities and hence increasing prestige for their work.
The aim to gain a reputation can also be viewed as a competitive motivation. Competition among programmers is an important stimulus in the motivation that leads people to get involved in the open source project. Finding the best solution or creating something entirely new gives intellectual gratification and intrinsic satisfaction. They participate, in a sense, to satisfy a demand for which there is no corresponding supply - in short to "fill an unfilled demand". (Green, 1999)

Another important factor that helps to explain the popularly of Linux is the availability of skill in people willing to develop software. Many programmers are current or previous computer science students that work with UNIX, while some are employees already familiar with this program. This fact significantly reduces the time investment in knowledge necessary for programming. Participating in the projects provides great information and ideas for research and graduate and undergraduate thesis. Also, having access to the source code represents an extraordinary way to learn and improve one's skills. The programmers can study the software very deeply and learn from following the questions and their answers. Studying the solutions to problems from the best programmers represents an immense learning opportunity. All these non-monetary benefits can be considered as ego gratification incentive.

As we saw earlier, contributions to open source software are motivated by the desire of programmers to show off their talents. In addition to the intellectual reputation gained by recognition and prestige among community peers (another important reason to participate), motivation comes from significant monetary benefits. Contributors can make money indirectly by displaying their abilities in the open source community. The reputation is in the public domain as a result of the system of credits. Gaining reputation and signaling talent gives programmers the chance to be noticed by software firms. The effort spent in contributing to the open source software can be turned into monetary benefits through employment by a commercial firm or through easier access to venture capital. Obviously the more famous a project is the greater are these advantages. Those can be considered, from an economic point of view, as career concern incentives. Lerner and Tirole[7] grouped the ego gratification incentives and career concern incentive in a unique category, the signaling incentives. As they point out, referring to the work of Holmström[8], those incentives are stronger the more visible the performance to the relevant audience is, the higher the impact of effort on the performance is and the more informative the performance about the talent is.

Eric Lee Green proposes a further element connected with monetary benefits that can be important to understanding the participation of programmers[9]. He suggests that there exists the motivation of advertising goods and services. Gaining reputation and recognition helps programmers to sell private service to those who require experts in order to sell help books and consultancy. He also underlines a special motivation connected with filling an unfilled demand: "there are people who like source code. They like to read source code with their breakfast. They debug source code on their lunch break. They hack new routines into source code after dinner time. Yet nobody is willing to sell them any source code to hack on. So they create their own."

An internet-based survey of 141 contributors to the Linux kernel (Hertel et al., 2002) shows very interesting results that help to define the characteristics of the programmers involved in the development of open source software. They utilize this data in analyzing the importance of different motivations that lead to the participation in the project. Some of these results of the survey are very interesting.

1. 67% of the programmers were full time professionals and 23% of them were students.
2. On average, they work 18.4 hours per week on the Linux development.
3. 20% of those participating received a salary for their Linux programming and work on a regular basis; 23% at least sometimes. The remaining subjects receive no salary for their effort. Their time on the Linus project is completely voluntary.
4. 38% of the developers' group could carry out the Linux-related programming during their regular working hours (although this did not imply that this work was part of their official job). The remaining 62% developed the software during their spare time.
5. 40% of the developed group spent less than 10% of their spare time working on Linux. 48% spent between 10% and 50% in programming and the remaining more than 50%.
6. 59% of the developed group indicated that they worked in a group, providing evidence that at least some spontaneous teamwork exists within the Linux kernel community.

[7] Lerner and Tirole 2002a
[8] Holmström 1999
[9] http://badtux.org/home/eric/editorial/economics/php

7. The average satisfaction score was 4.4 (standard deviation 0.7, scale range between 1 and 5), indicating that people were quite happy with the working process.

A prerequisite necessary in order for all these motivations to be sufficient to produce the development of Open Source software is trust. In this case, trust can be defined as the expectancy of the members that their effort will be reciprocated and not exploited by other members (interpersonal trust) and that the electronic support system works reliably (trust in the system)

So far, the culture and the motivations that drives Linux' contributors should be clear. Before analyzing the differences in the software's development, it is important to briefly point out Microsoft's motivations and culture: it is one of technical excellence focused on the development of high quality products that are to be sold in order to increase market share and maximum profits. In reference to Microsoft's working force, recruiters "want people who are not simply interested in programming for the sake of programming, but who seek personal challenges and enjoy shipping products into the marketplace, making money for themselves and the company"[10]. It seems clear that for Microsoft the development of a product is not an end, but just a means for accomplishing an end – to gain money. In this sense, the "community" inside Microsoft behaves according to the neoclassical maximization postulate.

From an economic perspective, having a mass market product which close down the boundaries and can work with little specialized knowledge from the user has enabled a high diffusion as well as a standardization of communication.

DIFFERENTS APPROCHES TO SOFTWARE DEVELOPMENT

In general, software development is very complicated and involves a substantial amount of experimentations and trial-and-error learning. This renders it a cumulative process where improvements are incremental rather than radical.

The Waterfall Model

Software development has traditionally been considered as a process in which we can differentiate three main parts:

Specification: that is, the definition and design of the product. During the specification phase, the main technical requisites and functions of the product are determined, as well as its architecture.

Development: during the development phase the ideas generated in the specification phase are implemented in code: developers create the constituting parts of the software product.

Integration and testing: once the different parts of the system have been created, they are integrated and tested in order to detect potential errors and incoherencies in their interaction.

[10] John Seabruck quoted in Cusumano and Shelby

What has been described so far constitutes what is called the “traditional” or “waterfall” linear model for software development. Although this methodology has been replaced by more modern practices, it is a useful conceptual framework for thinking about the basic tasks that should take place during software development.

The basic problem of the waterfall model is that it assumes perfect knowledge about the future evolution of the software project from its beginning. Given the complexity of software development, this certainty typically appears as an exception, not as the rule, most of the times. Unexpected integration, usability and bug problems that are difficult to solve without big changes in the structure that was designed at the beginning, are found at the latest stages of the project. The implementation of the specifications may become impossible because of unpredictable coding problems. Accurate scheduling becomes a heroic task and unreliable, malfunctioning and difficult to use products are at the end released.

The Spiral Model

The solution for many of these difficulties has been found in the “spiral model” – one that is followed by Microsoft. The spiral model acknowledges the uncertainty and complexity of software development by being less ambitious in its span and long-term planning. It consists of what it could be considered as an iterative version of the waterfall model taking smaller steps. To a certain extent, the boundaries between the different phases previously considered dissolves. Using Microsoft’s terminology, components are built, tested and integrated on a co-ordinated, “daily” base, in what has been called the “synch-and-stabilise” methodology. The specifications are much more flexible and abstract (basically defining what the product should be, according to the needs of the target market), allowing for a development phase in which problems are dealt with as they are found and the decision about trade-offs between different functions becomes simpler. After a certain amount of time the product is integrated and tested, and the encountered problems corrected, avoiding the kind of “infinite defects” situations found at the end of projects following the waterfall model. Testing is carried on almost as soon as the code is created in order to avoid bugs sliding into the evolved, incrementally constructed, future system. Usability is easier to build into the product from its inception through the exposure of users to the working releases that are being developed.

Still there are problems associated to the spiral model: they mainly have to do with the coordination and management of those groups in charge of the different parts that constitute the system. In order to function correctly, this methodology requires a very well synchronised evolution of the components that are integrated into the software program. And even though testing takes place on a more exhaustive and comprehensive basis than in the waterfall model, it is impossible to guarantee a “perfectly reliable” product given the wide scope of real situations and conditions in which it will be used (external complexity). Beta testing - that is, testing of the product by external users in real-life conditions before release - is a way of alleviating these problems, but not their final solution. There are still problems with schedules and quality of the product, especially in the case of Operating Systems, the most complex and difficult to develop software products.

The Open Source Model

The Open Source development model followed by Linux and other software programs such as Apache, the Pearl Programming Language or Sendmail is based on organisational arrangements that are very different from those described before. Open Source means that the code underlying the software program being distributed is available to anyone interested in it. Most software companies do not make their source code available, and consider it a part of their intellectual property, mainly for strategic reasons.

Open Source works on a decentralised basis, and the boundaries of what we could define as the “community of developers” are very loose. The development of products such as Linux is made by volunteers and could be described as follows: the manager of the “Linux Project” (Linus Torvalds) releases the different versions of the Linux OS and distributes them through the Internet. Those interested can delve in the source code of Linux and improve or test it, as well as create new features. The developers determine specifications and designs of new features and functions.

Any kind of change, improvement, correction or feedback regarding errors is submitted back to the “coordinating centre”, which is in charge of incorporating the improvements to the next release (that is, integrating the different elements contributed by the community into a new version of the product).This centre is also responsible for guiding the development of the project, by choosing the path of technological evolution from amongst all the possibilities explored by the community, and asking for additional effort in the areas where improvement is perceived as necessary.

The releasing policy in the case of Linux takes place on a different basis than in most software products. Instead of trying to release moderately reliable products by internal testing, Linux is released at a very fast pace, and reliability is achieved by the external testing in real conditions performed by users and the developer community. The numeration of the different releases indicates whether they are stable (in case they have been tested thoroughly and bugs have been detected and fixed) or not. Users can choose which of the versions to use, depending on their interests and objectives.

This model is based on a very special kind of licensing arrangement, the General Public License (GPL), which covers all Linux releases. That is, there is no possibility for appropriation of the development of Linux. Even when it is sold, users will have the right to access its source code and modify it, and redistribute or sell it. In this sense, Linux is public property.

Of course, the Open Source model is not devoid of problems: the decentralised, non-hierarchical organisational arrangement adopted as its development strategy can provoke what has been defined as “forking” (the apparition of different incompatible products) or mismatches in the assignation of the developers to tasks (maybe those more tedious but necessary pieces of work will not be performed by anybody). We should also take into account that the evolution and improvement of Open Source products depends on the will of the volunteer community. This can create significant uncertainties about the future evolution of the product.

The difference between Microsoft and Linux seems, by now, clear. Not only are their development models different, but also the motivations. And it is essential to take into account that these two aspects are not independent. Moreover, different objectives mean different constraints. As it will become clear later some of Microsoft's design and development decisions are constrained by the main objective of the firm, which is to maximise profits. Certain strategies that would allow for better quality products are neglected because they would loosen the control of the firm over its product and complementary markets, thereby reducing their potential profits.

THE MANAGEMENT OF COMPLEXITY

Having mentioned the interrelationship between motivations and potential strategies that can be adopted, it is important to focus on specific issues relating to the methodologies that each of the alternative paradigms being analysed employs to deal with the complexity of software development. I will point out where the constraints created by the motivations of the different communities restrict the adoption of those that seem to be more efficient development strategies.

Modularization

Modularization has been described as one of the more efficient methodologies for the building of complex structures composed of interconnected subsystems. It proposes the creation of independent subunits that then will be interconnected with clear and clean interfaces to finally give form to the larger construction. Through the modular process it is easier to create the product and maintain it (it is easier to determine what does what and to solve a problem by determining which module or linkage is malfunctioning). Modularization is certainly a way of improving software development. It makes design easier and also more efficient: building blocks with the same functions can be reused for different products. In a dynamic sense, modularization permits the easier substitution of different components and the enhancement of subsystems of the product. This reduces the complexity that would typically be dealt with when altering the product.

Linux is a very modular OS, based on design principles inherited from the first Unix OS. Modularity is essential for the Linux decentralised development model. According to Linus Torvalds (1999), the key to the success of open source is its modularity. Talking about Linux, he argues that "what is needed is as modular a system as possible. The open source model of development requires it so as not to have people working at the same point at the same time." Coordination is thus made possible by a sensible technical choice that in simple terms involves "keeping the kernel small and limiting to the minimum the number of interfaces and other restrictions to its future development".

On the other hand, although Microsoft has introduced in its development process modularization design techniques, it has on many occasions preferred to integrate products (that is, to link the source code of different functions instead of designing different modules for each of the functions). The reasons for this are performance (in this way the programs take up less memory and work together more effectively), and commercial ones: the use of integration makes bundling (selling different products in the same package) easier.

Decentralisation

As it has been noticed, organisational arrangements constitute an essential part of the development models followed inside technological paradigms. Hierarchy and decentralisation constitute different, although in most occasions complementary, kinds of organisational settings. Hierarchical organisations are based on a top-down approach to management: information (in the form of "commands") flows down the hierarchical chain to those subordinated, assigning them to different tasks. In the case of decentralisation, semi-independent units work in a more autonomous way (albeit subject to a certain hierarchical control in most occasions, that tries to co-ordinate the activities of the independent productive "cells"). The determination of whether or not decentralisation is possible within an organisation is, to a great extent, determined by the technological nature of the development process: certain products allow for a greater degree for decentralisation than others. In the last two decades there has been a strong surge towards decentralisation and redesigning of processes in order to allow workers for more independence and creativity in the performance of their productive activities. Some reasons for this are as follows:

Through decentralisation, it is easier to determine the output for each of the working units. This makes management easier (in a sense, the advantages of decentralisation are similar to those of modularity: it becomes easier to determine what is going wrong and where) and facilitate the linking of rewards to performance, providing incentives for workers to increase their effort, and therefore helping to solve the typical problems of principal-agent found in most firms.

Through decentralisation "informational responsibilities" are more efficiently distributed across the firm. The manager is, to a certain extent, relieved of the burden of coordinating the productive process to its finest degree (for what an enormous amount of information is required for this). Workers have a greater opportunity to take part in the decision-making process in the areas in which they are most knowledgeable.

In the same way, process innovation is favoured: there is no need for redesigning enormous and very complex production processes in order to introduce innovations. Small changes in the operating units can increase their performance without affecting the rest of the productive system. Workers, who are better informed about what they are doing, can introduce small-scale innovations. The human capital of the whole organisation is exploited in a more efficient and comprehensive way.

Microsoft is one of the organisations that, as Brooks signals, has developed a mentality of making “large teams work like small teams”[11]. However, given the need for integration (already mentioned) and for a certain degree of control over schedules and interactions between components in a less modularised product, decentralisation cannot be adopted in Microsoft to the degree observed in the case of Linux. The modularity of the Linux OS makes decentralised work easier: there is little need for co-ordination among a great number of developers localised all around the world. This decentralization has became possible because of the expansion of the Internet. The heterogeneity of these developers also provides Linux with what it might be called a “greater pool” for the exploration of the space design, which increases the innovative potential of this system: The open nature of Linux makes it easier for many different people to examine its code and consider new ways of improving it. Given the tacit, personal components of knowledge (very present in the case, for example, of software coding), this system allows for very diverse perspectives and original approaches to be channelled in the development of Linux[12].

Decentralisation makes it possible for literally hundreds of developers to concentrate on the same piece of code and explore different ways of improving and debugging it without an organisational hierarchy that is concerned with the “allocation of human resources”. From those suggestions, the best ones can be chosen by the centre – those in charge of integrating the system. Another advantage of decentralisation is that it avoids the problem of having to find a consensus amongst different projects inside an organisation. The independence between the different development actors makes it possible for everyone of them to act in its best interests, that is, the development of high quality components and functions.

Because of this same reason, and taking into account that Linux developers tend to be Linux users as well, we will find that decentralisation increases the diversity of versions for the system (different users adapt the system to their needs). This is a very straightforward way of integrating users into the design of software.

The main downfall associated with this kind of extreme decentralisation that is adopted in the creation of Linux is a potential loss of control over the development process. In the case of Linux, there can be no direct assigning of manpower to the solution of specific problems. Linux developers “assign themselves” to the tasks that they think they should be performing. Although in some senses this constitutes an advantage (self-assignment may be an easy way to match skills with tasks), on the other hand tricky and less exciting – albeit necessary – activities may be neglected.

Communication and Control

Communication and control of activities are deeply linked to the hierarchical or decentralised nature of an organisation. Management consists of a transmission of information that tries to influence an actor so that he/she performs a task in a way that is considered to be more efficient, given the objectives of the organisation. If we consider this definition of management, we could say

[11] Brooks 1995
[12] Raymond 1999

that in Linux there is no management at all. In the case of Linux the only flow of information that goes top-down is the release of new versions as a set of “information”. The feedback that the centre obtains makes up the actual components that will be integrated into the next release. Moreover, the programmers involved in Open Source explicitly use comparisons about the evolution of alternatives in terms of selecting the best and weeding out of the worst.

In the case of more formal management, there are decisions at the top which are transmitted to the bottom and then feed back from the bottom to the top, reporting on the results of the activities previously “ordered”. The need for information processing will be stronger in this kind of arrangement: the manager needs to determine what is needed and then ask his/her subordinates to do it. Then the manager obtains information about the results. In contrast, in the case of Open Source, the “manager” just tells the community what the situation is and the community gives him/her the potential answers from which they should choose.

The structure that we could more clearly associate to Linux has, therefore, some important advantages in the dynamic and complex world of software development. First, it becomes easier to avoid organisational inertia and adapt more flexibly to changes in the environment. Decisions come from the bottom – where more knowledge will be accumulated – that present a more accurate perception (and conscience of the importance) of potential future alterations in the competitive, technical and social setting.

Another important issue is that the Linux organisational arrangement increases this paradigm’s innovative potential. Lawrence Lessig argues that Microsoft, by keeping its source code secret and organising research and development on a hierarchical basis (it does not matter that the corporation is to a certain extent decentralised; still, basic decisions about allocation of resources and directions of research are made at the high levels of management), it can block those innovations that threaten its power or may cannibalise the sales of older products. After all, as I have said before, Microsoft’s main objective is not to produce a maximum quality product but to maximise profits. On the other hand, there is no management in Linux that can behave strategically in order to decide which innovations are to be developed an, in some extent, adopted. The nature of Open Source is, in a sense, quite democratic, and its leaders should take those decisions that are considered by the community to be the right ones.

Given that the objective is not profit-maximisation, the leaders of the movement do not have incentives to behave in ways that contradict the maximisation of quality objectives of the community.

This special characteristic of the Open Source movement – that is, the absence of a top-down management and the subordination of the “centre” that integrates the product to the will of the developing community – creates, on the other hand, the potential problem of forking, illustrated through the following example: Imagine a case in which two different ways of implementing a solution into the next release of Linux (A and B) are submitted to the centre. The centre chooses A because its members think that it is better, but a part of the community does not agree and instead chooses B, starting a new project using B as a solution. Linux will be forked, and the developer community split. Two different products will have been created, with potential incompatibility problems. Since the Linux model relies on the work of that volunteer community, forking would considerably weaken the strength and capacity of product improvements, with the network dynamics already mentioned creating a vicious circle of less developers, less quality, and therefore – again – less developers… Forking can be caused by disagreements over technical issues, and its avoidance relies, to a great extent, on the capacity of the centre of the community to choose the right decisions in what refers to technical matters. Until now, Linus Torvalds and his lieutenants seem to have been successful at this task.

Testing

Testing the software is probably the most important aspect in which the two approaches differ. Before analyzing the technical aspects, the economic relevance of testing has to be taken into account: complexity ensures that most of the cost of software arises from testing, debugging and customer maintenance, not from the original design and coding. One study found that testing, debugging and maintenance account for 82% of the cost of the software[13]

It has already been mentioned that testing inside a firm is unable to discover all the problems and bugs hidden within a program's code because of the impossibility of considering all of the potential scenarios in which this program will be used. In order to alleviate this difficulty, most software organisations – including Microsoft –tend to recur to Beta testing strategies (the test of the product by external users prior to release). In the case of Linux, the testing strategy adopted is quite different: products are mainly tested by users once they are released, in real conditions. Users can either solve the problems that they find and submit "patches" (corrected portions of code) to the centre, or provide feedback about them, signalling therefore where other developers should focus their debugging efforts. This is made possible by the availability of the source code and the modularity of Linux (that makes it possible to change some parts of the product without affecting the rest).

In a way, Linux users become testers of the product. This makes the debugging and improving of the product much faster and comprehensive. As Raymond states, "given enough eyeballs, all bugs are shallow"[14]. The potential problems associated with this approach – that is, uncertainty about the degree of reliability of a product at any given moment in time – are solved through the already described "two-track releasing approach", with the simultaneous availability of stable and experimental versions from which users can choose, given their interests and objectives.

Another fundamental advantage of Open Source, is that developers are not subject to the pressures that are induced in software houses by corporate announcements of more updated and efficient releases.

[13] Cusumano 1995
[14] Raymond 1999

LINUX’S RELATED ASPECTS

When we consider the Linux operating system, we need to take into account the presence of aspects that have very little to do with the creative work: programming; the development of graphical interfaces, the compilations of technical manuals, the online support in newsgroups and so on. Although these activities display a low level of innovation, they are fundamental for the adoptions of Open Source software. The spread across society depends on a cumulative series of little incremental innovation displaying, in many cases, of no intrinsic technological interest.

Another problem that emerges when analyzing the intrinsic characteristic of Linux is connected with its modular nature; for users, the modularity of the operating system poses problems. Getting a working operating system requires one to download all kinds of files. These files are the source code. They have to be compiled into working computer code. For many users, this requires more expertise or time than they might have available.

The solution to these problems seems to have been found by companies and software houses that base their activities on the commercialization of Open Source software; more specifically, they sell a “package” that, in addition to the free software, includes consultancy, manuals, maintenance, updating and training. This business model is widely accepted in open source communities and it is characterized by a detailed regulation presenting innovative legal solutions which address the relation between free and on-payment services. Red Hat, for example, is probably the most famous company of this sort that puts together and compiles the source code of the Linux operating system – obtaining a program ready for installation, together with a set of accessory applications and complete documentation, which is distributed on a CD at less than 100 dollars. What these Linux distribution companies do is to create economic value by packaging the Linux goods and services into a somewhat more standardized product. They also provided a focus for corporate users, who want some company to guarantee stability and provide technical support. Despite a discussion of market distribution, it should be noted, however, that the price between the two alternative operating systems is still dramatically different.

Besides playing an important role in the diffusion process, those companies simultaneously solve two problems: first they secure for potential adopters a wide range of complementary services, which are considered fundamental for reliability and usability of the software. Second, they provide monetary incentives for the development of “non-interesting” activities, which do not fit the typical motivations of the open source community. Moreover, distribution through the market can thus reduce other (latent or perceived) costs, even if it is cost to obtain the software package. In addition, it gives legitimacy as a “real” alternative

This is a fundamental point for two reasons: on the one hand, the tendency of open Source programmes to become more user-friendly enables their diffusion in increasingly broad bands of the population. On the other hand, 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 case of software the word “free” has the meaning of “free speech” and not “free beer”[15]

Usefulness and attractiveness to users is related to availability of complementary hardware and software. The usefulness depends to some extent on the availability and quality of other software programs as well as the types of hardware upon which it can be run. The word “attractiveness” highlights that the choice of an operating system depends not only on the actual users who have already chosen it, but also on potential users which might chose it.

Other types of firms – such as database or hardware vendors – contribute to further use and development of Linux by making their software or hardware compatible. Such compatibility and such recognition of the technical value of this operating system, in turn, give the core software a greater range of users as well as credibility and legitimacy. Both of these effects attract additional user of Linux, both for individual PCs and corporate network server. Thus, firms become increasingly important for software development of Linux.

In the case of Linux, as more users chose it, this stimulates programmers, hardware companies, applications companies, and so forth to develop appropriate goods and services around the operating system that might interest different types of actual and potential users.

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 the risk of having to change applications and platforms, incurring high switching costs for infrastructures and personal training.

NETWORK EXTERNALITIES

From the point of view of demand, network externality is an important feature that needs to be analyzed. Following the definition of Katz and Shapiro, “the utility that a user derives from consumption of the good increases with the number of other agents consuming the good” (Katz and Shapiro 1985). Externalities can be direct, indirect or deriving from complementary services. Direct externalities derive from individual physical membership of a network (the classic example is file exchange). Indirect network externality characterizes the so-called hardware-software paradigm in which two or more complementary goods are joined together forming a system that works only thanks to the contribution of all its components. The positive indirect externality is given by the fact that the complementary good becomes more quickly economical and available with the growth of the market in which the good is compatible with it. The third source of externality regarding complementarity arises from the idea that the quality and the quantity of services associated with the good is dependent on the number of units of the good that have been sold.

Dalle and at. (2005) refer to local interactions to explain the dissemination of the Linux system in place of Windows. 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.

[15] Raymond 1999

The characteristics of such a network vary according to whether the nodes are represented by Open Source software or by those using a commercial one. A widespread phenomenon amongst the former is in fact the so-called “advocacy” – theorized by leading members of the Open Source movement. This is a form of one-to-one marketing whereby the users of Open Source programs 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 diffusion process of Open Source software seems then to satisfy the hypothesis at the heart of the existence of Critical Masses in the spread of new technology, which permits an alteration of standards. This effect happens when certain variables characterizing a process rise above a given threshold. The phenomenon then 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. The emergence of the free software as a new production paradigm therefore, does not necessarily mean the end of proprietary software but the possibility of a new equilibrium in which the two paradigms are going to compete.

Another important aspect is that the diffusion of the Open Source software had radically different dynamics depending on the presence of the first mover advantage. In the sector of client-side operating systems, where Microsoft was the incumbent when Linux appeared, the market share of Windows is over 75% on most local markets. On the contrary, in the Web server sector, where the technology became established later and Microsoft had no consolidate advantage, the Open source system Apache is the incontestable leader. This suggests that whoever has the first move advantage and quickly aggregates the network externality effect is in the best position for dominating the market.

CONCLUSIONS

The productions of opens source software can be considered an example in which not only monetary and selfish motivations lead people’s behavior. Beside the possibility of signaling one’s own abilities offered by this new production system, altruism and satisfaction derived by belonging in the Open Source Community constitute a fundamental aspect in programmer's motivations.

In addition, altruism and social motivations, summarized by the idea that information (in this specific case the source code) should be freely available to everybody, contribute also to explain the participation in those project motivated by the strong opposition against private software’s firms and its leaders.

The particular motivations of agents and the rewards mechanism (both intrinsic and monetary) are such that programmers are very efficiently organized in a multidimensional network that includes not only programmers, but also user and tester, and often these roles interchanges. At a close view seems also that this organization is also able to produce software that seems of better quality that the proprietary one.

The presence in the scenario of commercial companies that make profit with Linux plus the networks externalities that Open Source Software create are other two important keys to understand the success and the diffusion of the Linux Operating System.

References

Antonelli, C., (2001): “Economics of Innovation and New Technology”, Taylor and Francis Journal, Vol. 11 pp 127-164

Arrow, K., (1994): “Methodological Individualism and Social Knowledge”, American economic Review, 84 (2), pp 1-9

Bassanini, A. and G. Dosi, (2000): “Heterogeneous Agents, Complementariness, and Diffusion. Do Increasing Returns Imply Convergence to International Technological Monopolies?” LEM Working Paper

Bensen S. M. and J Farrell, (1994): “Choosing How to Compete: Strategies and Tactics in Standardization”, Journal of Economic Perspectives, Vol. 8(2), pp 117-131

Berzoukov, N.(1999): “A Second Look at the Cathedral and the Bazaar”, First Monday 4

Berzoukov, N.(1999): “Open Source Software as a Special Type of Academic Research (Critique of Vulgar Raymondianism)”, First Monday 4

Bonaccorsi, A. and C Rossi (2002): “Why Open Source Software can Succeed?”, LEM Working Paper

Bonaccorsi, A. and S. Rossetto (1999): “Modularity in Software Development. A Real Option Approach”, International Journal of Software Development, Summer

Brooks, F., (1995):”The Mythical Man-Month” Reading, Massachusetts, Addison-Wesley

Cusumano, M. and R. W. Selby (1995): “Microsoft Secrets: How the World’s Most Powerful Software Company Creates Technology, Shapes Markets and Manage People”, New York: Simon and Schuster

Cusumano, Michael A. and David B. Yoffie (1998): “Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft”. New York: The Free Press

Dalle, J. M., David, P. A., Rishab D., Ghosh A. and W. E. Steinmueller (2005): “Advancing Economic Research on the Free and Open Source Software Mode of Production”, in Bulging Our Digital Future, Wynants & Cornelis, VUB Press, Bruxelles, Belgium

Dempsey, B. J., Weiss, B., Jones, D. and J. Greenberg (1999): “A Quantitative Profile of a Community of Open Source Linux Developers”, University of North Carolina at Chapel Hill, School of Information and Library Science, Technical Report TR-1999-05

Floyd, C., Züllighoven, H., Budde, R. and R. Keil-Slawik (editors) (1992): “Software Development and Reality Construction”, Berlin Springer-Verlag

Ghosh, R. and V. V. Pekash (2001): “The Orbiten Free Software Survey”, First Monday

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

Green, L. E., (1999): “Economics of Open Source Software” 1999
http://www.badtux.org/editorial/economics.html

Hars, A. and S. Ou (2002): “Working for Free? Motivations for Participating in Open-Source Projects”, International Journal of Electronic Commerce

Hermann, G., Hertel S. and S Nieder (2001): “Motivations of Software Developers in Open Source Software”, University of Kiel Working Paper

Hertel, G., Dater, C. and U. Konrad (forthcoming), (2002): “Motivations gains in Computer-supported Teams”, Journal of Applied Social Psychology

Himanen, P. (2001):”The Hacker Ethic and the Spirit of the Information Age”, London: Secker & Warburg
http://archive.salon.com/21st/feature/1998/05/cov_12feature.html

Holmström, B. (1999):”Managerial Incentive Problems: A Dynamic Perspective”, Review of Economic Studies, Vol. 66, pp 169-182

Hurbeman, B. and C. Loch (2002):”A Punctuated Equilibrium Model of Technology Diffusion”, Xerox Palo Alto Research Centre, Palo Alto

Jarvenpaa, S. L. and D. E. Leidner (1999): “Communication and Trust in Global Virual Teams”, Organization Science, Vol. 10, pp 791-815

Katz, M. and C. Shapiro (1985),:“Network Externalities, Competition, and Compatibility”, American Economic Review, 1985, Vol. 75, No. 3, pp 424-440

Katz, M. and C. Shapiro (1986): “Technological Adaptation in the Presence of Network Externalities”, Journal of Political Economy, Vol. 94, No. 4, pp 822-841

Koch, S. and G. Schneider (2002): “Effort, co-operation and co-ordination in an Open Source Software Project”, GNOME, Information System Journal, Vol. 12, pp 27-42

Lakhani, K. and E. von Hippel (2000):“How open Source Software works: “free” user-to-user assistance”, MIT Sloan School of Management Working Paper #4117 (May)

Lakhani, K. R., Wolf, B., Bates, J. and C. DiBona (2002): “The Boston Consulting Group Hacker Survey” http:/www.bcg.com/opensource/BCGHackerSurveyOSCON24July02v073.pdf

Langlois, R. N. (forthcoming) “Modularity in Technology and Organization”, Journal of Economic Behavior and Organization

Lerner, J. and J. Tirole (2001): “The Open Source Movement: Key Research Questions”, European Economic Review, 2001, No. 45, pp 819-826

Lerner, J. and J. Tirole (2002a): “Some Simple Economics of Open Source”, Journal of Industrial Economics, Vol. 52, pp 197-234

Lerner, J and J. Tirole (2002b): “The Scope of Open Source Licensing”, Harvard Business School, Working Paper, Boston, MA

Lessig, L. (2004): ”Future of Ideas -- Microsoft Reader eBooks” at
http://www.ebookmall.com/ebook/114228-ebook.htm

Liebovitch, E. (1999): “The Business Case for Linux”, IEEE Software, 16(1), pp 40-44

Manfredi, P., Bonaccorsi, A. and A. Secchi (2000): “Hetherogeneity in New Product Diffusion”, LEM Working Paper

Mateos Garcia, J.(2001):”Can Information be free?: Motives and Evolution of the “Information should be Free” Principle”, SPRU, Spring Term Paper

Mauss, M.(1959):”The Gift. The Form and the Reason for Exchange in Archaic Societies”, Routledge London

Moody, G. (2001):”Rebel Code. How Linus Torvalds, Linux and the Open Source Movement are Outmastering Microsoft”, London: Allen Lane, the Penguin Press

O’Reilly, T (1999). “Lessons from Open Source Software Development” Communications of the ACM 41 (4), pp 33-37

Osterhol, M., J Frost and A. Weibel (2002): “Solvin Social Dilemmas: The Dynamics of Motivations in the Theory of the Firm. University of Zuricg W.P., Zurich Switzerland

Pedersen, Soren Thing (2001): “Open Source and the Network Society”. http://opensource.mit.edu/papers/pedersen.pdf

Raymond, Eric Steven (1999): “The Cathedral and the Bazaar” at http://tuxedo.org/~esr/writings/cathedral-bazar/cathedral-bazar

Rosenberg, D. K. (2000):”Open Source: The Unautorized White Papers” Foster City, CA: M&T Books

Shapiro, C. and Varian, H. (1999): “Information Rules: A Strategic Guide to the Network Economy”, Boston, Massachusetts: Harvard Business School Press

Stallman, R. M. (1994):“The GNU Manifesto”, http://www.gnu.org/gnu/manifesto

The Economist (1999): “Computer programming. Hackers Rule”, 20 February 1999

The Economist (1998): “Free software. Red Hat trick” 3 October 1998

The Economist (1998): “Software. Revenge of the hacker” 11 July 1998

The Economist (1998): “The revenge of the Hackers”, 9 July 1998

Toch, H. (1965):”The Social Psychology of Social Movements”, Indianapolis: Bobbs-Merrill.

Torvalds, L.(1999): “The Linux Edge” in DiBona, C., Ockman, S. and M. Stone: “Open Source: Voices from the Open Source Revolution”, Beijing, O’Reilly and Associates

Tuomi, I. (2001)“Internet, Innovation, and Open Source: Actors in a Network”, First Monday

Ullman, E. (1997):” The dumbing-down of programming”, at http://archive.salon.com/21st/feature/1998/05/13feature.html

Ullman, E. (1997): “Close to the machine: Technophilia and its discontents”, City Lights, New York, NY

Wellman, B.: “An Electronic Group is Virtually a Social Network”, in S. Kiesler (Editor) (1997), Culture of the Internet (Erlbaum, Mahwah), pp 179-208

Witt, U. (1997):”Lock-in vs. Critical Masses. Industrial Changes under Network Externalities”, International Journal of Industrial Organization, No. 15, pp 753-772