MSc Dissertation

INNOVATING WITHOUT MONEY:

Linux and the Open Source Paradigm as an Alternative to Commercial Software Development

Juan Mateos Garcia

August 2001

Submitted in partial fulfillment of the requirements for the degree of Masters of Science in Science and Technology Policy

Supervised by Professor W. Edward Steinmueller

SPRU - Science and Technology Policy Research
University of Sussex

SUMMARY

This dissertation analyses two different paradigms used for the development of a software product, Operating Systems. The targets of research are two groups that operate in very different ways, Microsoft Corporation and the Linux Community. Through the observation of the strategies and methodologies used by these actors in their work, and taking into account the constraints to which they are subject, assesses, from a dynamic perspective, the relative strengths and weaknesses of their competing paradigms. In the dissertation I will analyse the efficiency of the development processes that they have adopted for the design, improvement and enhancement of their products. A theoretical model based on Giovanni Dosi´s "Technological Paradigms" framework, incorporating institutional, industrial, social and cultural aspects, is constructed and specifically adapted to the software industry case. The *private management of technology* issues considered during the analysis of the development processes inside different organisations are linked to questions having to do with the evolution of high technology, networked, markets. This analysis lends to assessment of some rationales and potential strategies for public intervention in the Operating Systems industry, taking into account some relevant concerns that have been raised about competitive issues and dominant position in this market, one of strategic importance for the future development of the Information Society.

"Perfect scepticism makes perfect freedom possible…One can almost say that when belief in an object or a cause comes to an end, this object or cause returns to chaos and becomes common property. But perhaps it is necessary to have resolutely, forcibly, produced chaos and thus a complete withdrawal from faith before an entirely new edifice can be built on a changed basis of belief"

Hugo Ball, German Dadaist (1886-1927)[1]

"Money has nothing to do with the results. Money kills creativity. You can do it without money, you should just think how"

John Cassavettes, Film director (1929-1989).

[1] Quoted in Auster (1997) p. 54.

ACKNOWLEDGEMENTS

I would like to thank, first of all, professor Santiago López García, at the Universidad of Salamanca, Spain, for introducing me into the evolutionary economics paradigm, and recommending me the SPRU as a possibility for further development in my academic career. Many other persons there, such as David Anisi Alameda, Miguel Ángel Malo Ocaña or Crístina Pita are also responsible for the humble amount of human capital that I have been able to accumulate during the years of my degree. The economics department at the Universidad de Salamanca is an example of how enthusiasm and intellectual courage can make it possible to create even in the most arid environments.

Edward Steinmueller at SPRU is the person that I needed for the supervision of this dissertation. His open spirit has stimulated me into new directions and taught me how to look at problems from very different angles, while his logical discipline and critical mind have restrained me from diffusing my effort, loosening my grip and losing my focus when dealing with such an amorphous matter as reality.

My friends at Brighton also deserve enormous credit for their support during my bad days. I do not know if I deserve the friendship of such beautiful, honest people, but I am sure I was lucky in finding them. I know I am posing this question in the wrong terms (as I do most of the times), but please forgive me. I love you Erika, Juan, Mario, Rolando! Arriba Trafalmagore!

My family is responsible for all I am and all I have done. Without them, nothing would make sense.

I hope all this is worth the time and resources you have invested in me.

INDEX

INTRODUCTION

This dissertation is focused on the software industry and its dynamics, both internal and external. When I say internal, I refer to the product development processes that take place inside different firms. The external aspects have to do with the factors that affect competition in the industry as well as its links with broader society.

One point that must be made before starting is the fact that although I have used previously the word "firm", this term is not very precise when taking into account the nature and behaviour of the actors whose actions constitute the object of analysis. My main concern has to do with the development processes followed by two different "organisations", Microsoft and the Linux community. The fact is that the second group has challenged many assumptions about the ways in which a software product can be created, improved and distributed, raising some interesting questions about incentives, motivations and ways of working in the Information Society. As we will see later, the words "technical culture" or simply "community" seem more suitable when considering these groups.

My second concern has to do with the nature of competition in the software industry, and the processes through which "efficiency" and other socioeconomic and institutional factors influence its dynamics. The way in which operating systems, the software products on whose development I will focus, are put to use in networked environments, makes it necessary to consider certain external aspects that turn the study of this industry into a specially interesting and complex subject. It, therefore, deals with market power and industrial structure, another set of issues that have also become especially relevant.

This dissertation is composed of two main, complementary parts, one theoretical and the other one empirical. A model that tries to show how the different ways in which products are designed and developed determine their quality, taking into account the specific characteristics and complexities of operating systems, is built. The model also considers how these internal aspects and results are linked to the external environment, and the processes through which this environment may affect the potential success or failure of the competing products. The empirical part consists of a case study, or more precisely, an assessment of the relative efficiencies with which the different development models followed by each of the competing groups are able to construct an operating system, taking into account the intrinsic difficulties of this task. It is, therefore, basically related to the first part of the theoretical model, that is, the internal dynamics inside the different communities devoted to the development of operating systems. By showing how this evidence lends support to the hypothesis that certain development methods are able to produce better quality products and putting it together with the second part of the theoretical model, having to do with the external factors that affect competition in the operating systems market, I will reach some interesting insights from which policy implications can be drawn.

This may seem a bit too abstract, but I ask the reader, especially if he or she is not an expert in the subject, to be patient. Given the complex nature of the issues I am going to be dealing with, if I am able to, at least, make some sense of the interactions, processes and present in the fast-changing, networked environment of the current software industry, and explain them in a systematic way, one of my main objectives will have been accomplished.

STRUCTURE

The dissertation is structured in five parts having to do with the different aspects of the world I am going to be focusing on and my objectives when analysing them. The divisions between the parts are mainly thematic, but also have to do with an explicit intent to make the work coherent and readable.

FIRST PART: RELEVANCE, CONCEPTS AND OBJECTIVES

In this first part I mention some of the important topics this dissertation deals with. I show the relevance of my work both in academic and policy (business and governmental) areas, as well as introduce some concepts and definitions that will be used later in the dissertation. I also pose the main questions that will be addressed during this work.

SECOND PART: THE THEORETICAL MODEL

Having shown the relevance of research on the topics mentioned, I will first, as an introduction to the main part of my work, present a summary of the theoretical model that I am going to develop and use, and draw a map of the logical road I will follow during my research.

Having sketched a broad picture of the model that I will use for my enquiry, I will proceed to delve into the details, literature and relevance of its different parts. Given the heterogeneity of the intellectual materials used during its construction, instead of starting with a theoretical framework from which the model will be developed later, I will present the broader aspects underlying each of the parts of the model, as well as the relevant literature, as these different sections are introduced. A different section will be used to deal with each of the conceptual blocks used to build the model.

THIRD PART: THE CASE STUDY

In this part I will tackle the empirical work, that, as I have mentioned will consist of a comparative assessment of the relative strengths and weaknesses of the different development models used by two different organisations, Microsoft Corporation and the Linux Community, in the production of operating systems (Windows and Linux, respectively). Methodological questions, as well as limitations of my approach will be discussed. This part will be linked to the "internal" dynamics of software development, that is, to the part of the theoretical model related with the development of a complex product inside an organisation.

FOURTH PART: STRUCTURE, INSTITUTIONS AND INTERVENTION

The "external" aspects of the software market mentioned in the theoretical model will be considered in this part. I will study how previous history and industrial and institutional structures, in conjunction with the network dynamics observable in software markets, may affect the results of competition between different products. I will illustrate briefly some of the complex interactions between quality of the product and other factors present in software markets, and specifically in Operating Systems, focusing on the competition between Linux and Microsoft Windows. I will show some reasons for concern about the future evolution of this sector, as well as some alternatives and spaces for public intervention.

FIFTH PART: CONCLUSIONS

In this last part some issues that have been left of the dissertation are mentioned and some questions for further research posed. I will also make some final considerations about this work.

1- RELEVANCE, CONCEPTS AND OBJECTIVES

In this first part I am going to present the issues that drove me to choose competition between different Operating Systems as the object of my research as well as some of the questions that I want to address during this work. This part can be considered as an introduction in which some of the main themes underlying my dissertation will be introduced. I will also use the following lines to provide the reader with some definitions and concepts that will be used later in the work.

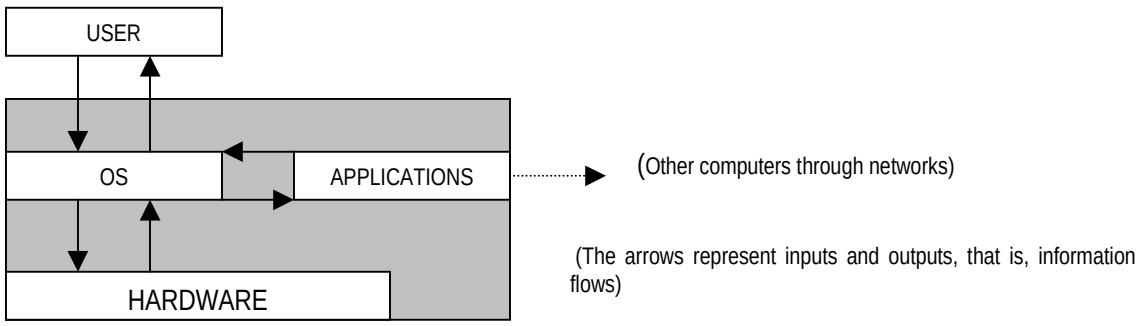
1.1- RELEVANCE

First of all I want to discuss some reasons that justify the necessity of research on the Operating Systems industry and its competitive dynamics. Some of the mentioned issues and topics will not be dealt with in depth during my dissertation, but I think should be addressed in the future if we are to make sense of the dynamics of networked markets and the evolution of technology in Information Society. Having previously presented a scheme of my work in very abstract terms, I want to show now the relevance of studying Linux, Microsoft, their development models and their interactions inside the software industry in general terms, not limiting myself to the scope of this particular dissertation. At the end of this section I will signal those topics in which my work is concentrated by posing some general questions to which I would like to suggest answers.

1.1.a) The Relevance of Operating Systems.

An Operating System (OS) consists of the "*Mechanisms, policies and procedures that support the controlled sharing of computer resources*"[2]. OSs can be understood as co-ordinators that manage and regulate the use of computing power stored in the hardware. They act as the main interface through which the user interacts with a computer and computers interact between themselves through networks. That is, an OS is the main structure that makes it possible for a user to communicate with its computer and with other networked users. As it can be seen in table 1, the OS also acts as a layer between hardware and applications (the software programs that users actually employ to perform more or less useful tasks).

TABLE 1- An scheme of a computer as the interaction between different elements



All communications between the user and the computing resources of its hardware are mediated by the Operating System. We could say that the OS coordinates the information flows between the different parts of the computer and between the computer and the external environment, including users and other computers.

[2] This definition is by Phil Enslow, quoted by Peter Salus in his history of Unix (Salus (1994) p. 22). For another simple definition of Operating System see Brown (1997) p. 89n.

Operating Systems are an essential part of any computer. They are the platforms where user interfaces, networking capacities, security, input-output transfer and many other basic functions and tasks are managed. In a somewhat rude way, Brown says that "(The OS*) precisely defines what (*a computer) *can and cannot achieve*".

The importance of OSs should not, therefore, be underestimated. They are one of the basic bricks with which information Society is being built. This brings us to the question of relevance of the research in their development process and the competitive dynamics of their market.

Quality and productivity:

Many authors have studied existing problems in the design and development of software products[3]. A software program is the instructions followed by a computer in the performance of a certain task. OSs are the most complex of these constructions, reaching in their size hundreds of millions of lines of code that control the interactions between different components of the computer, applications, hardware, networks and users, in the most diverse circumstances. Any error or inconsistency in that code (what is defined as a "bug") will generate problems. The design, development and testing of OSs are difficult tasks, and the finally released products are in many cases unreliable[4].

Being fundamental components of computers, OSs can become a bottleneck for productivity increases when there are problems with their quality, reliability or performance. Many authors have referred to these factors when trying to explain the productivity paradox observed in the case of Information Technologies[5]. According, for example, to Gibbs, one of the reasons for the small productivity increases that IT have brought to many industries is the fact that users have to waste important amount of time "futzing" with their computers due to errors in the software or slow performance[6]. OSs condition to which extent is the continuously growing computing power of micro-processors exploited and, by defining the interface between users and computers, also determine in a very important way the usability and usefulness of the latter. Therefore, it is very important to study where do the problems in software development come from, and which methods can be used to solve or alleviate them.

I will assess the relative efficiencies of different development methodologies in order to address the question of which organisational arrangements and working practices can be expected to be more useful for the creation of quality OS products, taking into account their inherent complexity. These questions are not only relevant in a business context, but also in a broader, social one, given the growing importance of electronic devices that rely on OS for their functioning.

Industry, innovation and competition

I have already said that OSs are a key technology in the development and exploitation of Information Technologies. As we know, innovation and improvement in a product are to a very important extent determined by the competitive structure of the markets Therefore, we should not only be concerned with the technical development of these systems inside organisations but also with the competition processes taking place between different products in the marketplace. Some special

[3] Brooks (1995) constitutes a very relevant summary of the many problems found during the development of software products.

[4] Although many methods and heuristics have been devised with the objective of improving the efficiency of software development processes, this activity, as we will see later, is still very problematic.

[5] The productivity paradox is a controversial question: it has to do with the fact that the huge investments in Information Technologies that have taken place during the last decades have not produced the expected increases in productivity. See Brynjolfsson (1993) and Landauer (1995) for discussions and data on this issue.

[6] Gibbs (1997), pp. 69-70. Landauer (op. cit. note 5) argues that software is the his "*candidate for the principal culprit*" of the productivity paradox (p. 125).

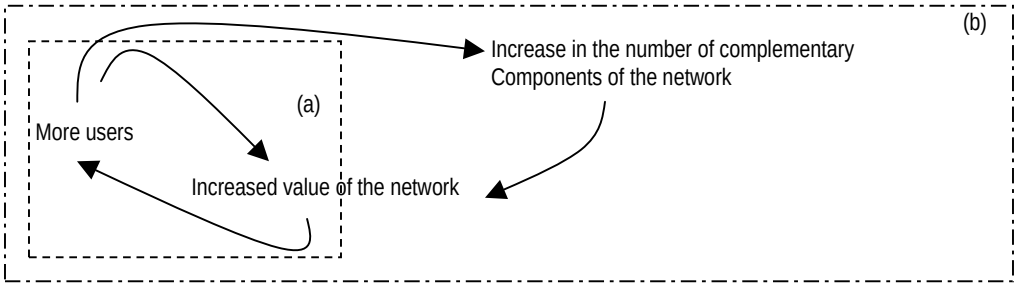
dynamics present in the OS market make it necessary to analyse it very carefully, taking into account not only economic, but also institutional and social factors. They will be considered in detail later, but I will mention some of them now.

-Network effects: computers exchange information in networks. This makes compatibility, that is, the ability to communicate between different systems an essential issue and brings to existence what has been called "network effects". According to Shapiro and Varian, these effects or externalities take place "*when the value of a product to one user depends on how many other users there are*"[7]. Network effects can create virtuous circles (positive feedback) in which the growth in the number of members of a network increases the value of the network and therefore provide incentives for further entrance into that network, starting dynamics that can lead to the eventual dominance of markets by one product. One way to avoid these dynamics of "tippy markets" in which the winner takes all is to create standards of compatibility so different products can coexist, communicate and compete in the same market. The problem of this kind of arrangements is that actually most agents have incentives to "betray" them in order to achieve the extraordinary profits derived from owning a standard and controlling the whole market.

-Market linkages and institutional structures: compatibility is also an essential factor to be taken into account when considering the issue of systemic integration. This is another factor that strengthens the importance of network effects. OSs are components of large systems that also include applications and hardware. The network effects activated in one market can start reactions that intensify the virtuous circle processes previously mentioned[8]. In table 2 we can observe a simple scheme of how do intra-market and inter-market network effects work.

We should also take into account that the development of useful, coherent systems depends to a large extent on co-ordination between different actors. We can observe in the computer industry how diverse institutional arrangements and inter-firm consortia are created in order to collaborate in the creation of integrated systems, in what as been labelled as the phenomenon of "coopetition". Although all these arrangements are necessary, they can turn sometimes be used in strategic ways to raise entry barriers that protect incumbent actors (that have already formed strong industrial networks) from competition.

TABLE 2- Network externalities as an intra-market and inter-market phenomenon.



Consider (a) as a network of users of an OS (that need compatibility in order to communicate). We observe intra-market network effects inside here. The positive feedback generated by the increase in the number of users of that network will also provide incentives for the production of complementary components and further increase the value of the network (we observe inter-market network effects in (b)).

[7] Shapiro and Varian (1999) p. 13.
[8] See Hill (1997) for an analysis of the incentives of establishing a de facto standard in "winner takes all" industries.

-Technology lock-in: When studying software markets, another issue that should be considered is that of customer lock-in to certain technologies. Lock-in can be roughly summarised as follows: "*Once you have chosen a technology or a format for keeping information, switching can be very expensive*"[9]. There exists a series of costs associated to switching from one technology to another, that can effectively tie a customer to the supplier of a certain technology. Some examples of these costs, especially relevant in the case of OSs, are the training costs necessary to learn how to use the new technology or those associated to changing the formats of already stored data. Technology lock-in is another way in which entry barriers for potential entrants in an existing market can be created.

All the previously mentioned factors make the study of the industrial dynamics in the case of OSs development and distribution an issue of public interest. Given the strategic character of OSs, and the special conditions in which they are distributed and adopted, research on the competitive dynamics that take place inside the industry becomes especially relevant. The concerns raised by the dominant position of Microsoft Corporation in this market, and the anti-trust lawsuit against it that has been taking place during the last decade, are examples of the relevance of understanding how does competition evolve inside high-technology, networked markets, and how does the industrial structure influence the rate and direction of technical progress.

I will analyse the structure and mechanisms through which competition takes place in OS markets with the aim of determining the need for public intervention and the potential forms it may adopt.

1.1.b) The relevance of Open Source

Having shown the importance of research on the development and market competition processes present in the OS sector, I will now refer to the reasons that make research on the behaviours and strategies adopted by Open Source Communities such as Linux, relevant.

The Open Source movement can be defined as a culture of individuals devoted to the free development and distribution of software. Open Source communities organise in collaborative networks that communicate through the Internet, avoiding any kind of barrier to the free flow of information between developers and users. While orthodox corporations usually retain the source code of their software "closed", and consider it as a "trade secret", open source communities freely disseminate the code which is the base of their products and ask anyone to delve into it and try to improve it[10]. Some of the most important actors organising projects that follow Open Source methodologies are Linus Torvalds, creator of the Linux Operating System (in whose development model I will focus during this dissertation), the Internet Engineering Task Force, in charge of the design and implementation of the standards that regulate the operation of the Internet, Brian Behlendorf, responsible for Apache, one of the leading Internet web servers, Larry Wall, creator of the Pearl programming language or Richard Stallman and the Free Software Foundation, developers of many of the tools, applications and programs used by the Linux OS.

These are just some examples of a Modus Operandi and a philosophy that until now have been found to be quite successful, challenging many existing assumptions about the motivations of people creating and developing technological artefacts and the optimal ways of co-ordinating the

[9] Shapiro and Varian (op. cit. note 7) p. 11

[10] The source code is the set of instructions written in a programming language that constitute the basis of any software program. For a history of the Open Source movement see, for example, Moody (2001).

construction of complex systems[11]. Questions about why do Open Source contributors behave the way they do, in a seemingly altruistic way, and how are they able to manage the development of complex software programs in a decentralised, non-hierarchical fashion, are important subjects for the social sciences[12]. During my dissertation I will deal mainly with the second question, that is, the one related to the strategies of software development through the Internet adopted by Open Source groups, although at some points I will also touch the first one, having to do with the possible motivations and incentives of members of Open Source communities . In my opinion, the analysis of the strategies and methodologies developed by the Open Source movement can teach us a great deal about the management of complexity and the incentives followed by individuals coming from technical and scientific cultures.

1.1.c) Theoretical relevance: A change in the techno-economic paradigm?

A techno-economic paradigm is, according to Christopher Freeman, a "*set of common-sense guidelines for technological and investment decisions as pervasive new technologies mature*"[13]. That is, it constitutes a dominant set of technologies that affect the productive structure of the whole economy. A change in the techno-economic paradigm could be understood as the diffusion of clusters of radical innovations that extend through society causing organisational, industrial, institutional and cultural change[14]. According to Christopher Freeman and Carlota Pérez, the core technologies of the currently dominating techno-economic paradigm are microelectronics and telecommunications, the Internet, that could be understood as one of the radical innovations mentioned, being one of its manifestations[15]. Previously I said that Open Source groups use this web as one of the main tools for the co-ordination of their productive activities. According to Moody, "*none of* (Linux) *rapid-fire development and distribution would have been possible without the Internet. The rise and success of Linux is bound up with the Internet at every stage*" [16].

I will show later that the organisational arrangements through which the Linux community performs its work are very different from dominant productive structures. Most of its members co-operate through the Internet on a decentralised fashion, with almost no hierarchies[17]. It seems clear that without the Internet , the development model being used by the Linux Community would not be possible, and we may ask whether this model appears as an organisational innovation, a change determined by the diffusion and assimilation of the technologies that constitute the new techno-economic paradigm[18].

The competition between Linux, representing a radical departure from traditional ways of organising production and Microsoft, that follows more orthodox methods, brings to us the following

---

[11] During the last years some "for-profit" corporations such as Netscape (with the creation of Mozilla.org), Macintosh (with the Darwin OS) or even Microsoft (through the Shared Source initiative) have tried to imitate the Open Source strategy.

[12] Mansell and Steinmueller have already pointed out the relevance of research on the topic of Virtual Communities (groups of shared interests that organise their activities through the Internet), inside which I would doubtlessly place the community of Linux developers. They set these communities in contrast with other two groups of actors, Incumbents and Insurgents, whose main objective is the increase of profits and dominance of markets through diverse strategies. For more information see Mansell and Steinmueller (2000), specially Ch. 1.

[13] Freeman (1992) p. 198

[14] Ibid pp. 132-139.

[15] Ibid pp. 200.

[16] Moody (op. cit. n 10) p. 68.

[17] The possibility of using a transaction cost framework, following Coase (1937), to analyse how new technologies alter the dimensions and structure of the firm becomes a possibility. The problem with doing this are the assumptions (or limitations) embedded in Coase´s theory: it only considers two potential ways of organising exchange, inside the firm or through markets. As I have mentioned before, Linux does not fit any of the two models.

[18] Kelly (1998) points out the existing uncertainty about the adecuate productive and business models to be adopted in the "network economy".

quote by Freeman, in which he refers to the obstacles (inertias) that stem from an old paradigm trying to protect it from change:

> "The massive externalities created to favour the diffusion and generalisation of an existing paradigm act as a powerful deterrent to change for a prolonged period. Arthur has demonstrated just how powerful these containment mechanisms, and David has shown how they can be influenced by technical standards. The existing stock and availability of skills, capital equipment, components, markets and enabling institutions is an extremely powerful combination favouring the already established techno-economic paradigm" (Freeman op. cit. n 13, pp. 135-136)[19]

Maybe Linux could be considered as an organisational innovation that represents an example of the diffusion of the technologies associated to the new techno-economic paradigm. By drawing the opposition between Microsoft and Linux in the hypothetical terms posed previously, I mean to say that the comparison between both groups can and should take place considering broader, systemic issues, and that very interesting conclusions in what refers to the extrapolation of future trends in certain sectors could be drawn from this study. I think that studying the dynamics through which Linux appeared and has extended could teach us something about how the pervasive influence of the new techno-economic paradigm is affecting ways of working and organisational settings in Information society.

This latter question will not be explicitly addressed during my dissertation, but underlies it. I think that it represents a very interesting topic for further research in the future. The adequacy of the "techno-economic paradigms" framework for the study of the kind of developments I will be analysing in this dissertation could be understood as a test for the theory. What I will draw, on the other hand, from the previous discussion, is the consideration of social and institutional issues (such as, for example, Intellectual Property regimes, as we will see later) when analysing the competition between the incumbent (Microsoft) and the new entrant (Linux). I have previously spoken about the "external" conditions that affect the competitive dynamics in OS markets, and defined briefly some concepts related to the economics of these markets. But I have not considered yet those broader social and institutional issues to which Freeman refers in the previous quote, and that may represent an obstacle to the diffusion of innovations. They will be brought forward later in the dissertation. Following Mansell and Steinmueller, as well as Social Construction of Technology theses, during my dissertation I will assume the need to consider social, economic and cultural issues that affect the diffusion of new technologies and the competition between existing ones[20]. Implicit and explicit considerations are made by actors when choosing among technological alternatives, and I want to examine some of the factors, not directly related to the quality of the product or the efficiency of its development process, that affect them.

---

[19] Before continuing, I will ask the reader's forgiveness for bringing these simplistic extremes into the discussion, and putting Microsoft in the field of what could be labelled as "organisational practices belonging to the old era". Of course, as we will see later, Microsoft working practices have evolved as the corporation has learned and grown, and we could say that they have set the benchmarks for most firms searching for profits in Information Society. I am not saying either that Linux represents a "radical" innovation in relationship to Windows, Microsoft´s product (In fact, Linux´ genealogy goes further ways than Windows).

[20] "*...the interactions between incumbents, insurgents, and virtual community constituencies is both a political and economic process. We have attempted to lay the foundations for a deeper consideration of these issues in which institutional, technological, social and economic influences shape the relative strengths of alternative strategies*" (Mansell and Steinmueller (op. cit. n. 12).

1.2- OBJECTIVES

The primary objective of this dissertation, and the one that motivates the use of several perspectives, is to provide reasoned answers to a series of questions that have to do with the likely evolution of the OS market when taking into account the relative efficiencies of the considered products (as determined by their development process). To account for this evolution, however, it is necessary to examine the issue of whether external factors, not related to the quality of the competing products, can end up determining their eventual success or demise in the OS market. Many of the issues I will end up dealing with have to do with power structures and strategic networks of actors in incumbent position that can behave in a collusive way in order to maintain and extend their control over certain markets. This is quite a controversial issue, as the reader will surely be already imagining the identities of those mysterious "incumbent actors". Across this dissertation, my aim will be to determine whether the analysed facts and behaviours justify policy intervention, and how can it take place. (For a scheme of the previous discussion see Table 3).

TABLE 3- Issues and questions

Subject	Relevance of research on…	Questions addressed
Open Source Movement	-Open Source motivations and methodologies	-Which paradigm is more efficient in the development of OSs? -Which other factors affect competition in the OS market. Through which channels?
Operating Systems	**-Efficiency of OS development methods** **-Competitive dynamics in OS markets**	
Macro evolution of technology and society	-Broad social and institutional factors and their influence in future developments. -Future trends and adaptability of observed phenomena to the theoretical framework	**Information for action** -Likely future developments in the OS market -Necessity and possibilities for regulation.

The bold issues are those that constitute the main objects of research of my dissertation. The underlined ones will be used as sources of necessary/auxiliary evidence when necessary.

2-THE THEORETICAL MODEL

In this part is where my actual research begins. Until now I have been referring to some relevant topics that will be addressed during my dissertation, and trying to outline the complexity of the world that I will be studying. Research in the field of science and technology policy should start by accepting and embracing that complexity of the world as one of the few methodological "realities" that the social scientist is sure to find. In order to make sense of that complexity, the use of theoretical models that introduce some analytical order in the world is needed. That is what I will try to do in this part of my dissertation.

2.1- INTRODUCTION

2.1 a) The Scheme

This dissertation is composed of two main parts, one theoretical and the other one empirical. The theoretical model that will be constructed in this part deals with the development of OSs, competition in the OS market and how historical, social and institutional factors affect them. I will base my analysis in the abstract concept of "technological paradigms" (defined following Giovanni Dosi). I will study how the different methodologies used by those inside the paradigm determine the attributes (quality) of their products, and how that quality interacts with other important factors as competition takes (or does not take) place in the marketplace. The empirical part of my enquiry (The Case Study) will be related to that first part, and devoted to a assessment of the relative strengths and weaknesses of Linux and Microsoft, considered as two different technological paradigms, in the development of OSs. This analysis will generate some insights in what refers to the organisational arrangements more suitable for the development of complex software products. But these are not the only potential results of my research. By linking the observed behaviour and results of different actors, with those broader issues explicitly considered in the theoretical model, I will, in Part 4, give some guidelines for policy research and action in the OS market.

In this part I will, therefore, proceed to the construction of the model that will be used as a basic framework in which both empirical and policy analysis will be strongly embedded. In order to avoid an excessive "specialisation" of the model on the industrial sector (Operating Systems) that concerns me, I will try to build it following a modular structure, using different components coming from different disciplines and schools of thought. As I do it I will mention those abstract aspects of each of the parts that could be considered in the analysis of other industries and then refer to specific details relevant for the study of the OSs sector.

2.1.b) A summary of the theoretical model

My model is composed of five sections: the first one is based on Giovanni Dosi´s "Technological Paradigms" framework, and considers how do technological innovation and development processes take place inside a technical culture. Since the possibilities and results of those activities are strongly determined by factors having to do with the nature of the product that is being developed and the needs that it aims to fulfil, another section will be devoted to the study of the environment in which Operating Systems are built and used. The theoretical framework underlying this second section will be based on literature about the development of software products.

Another issue that I will touch upon in this block will be how the legal environment , and especially Intellectual Property regimes, affects the motivations and incentives of the analysed actors.

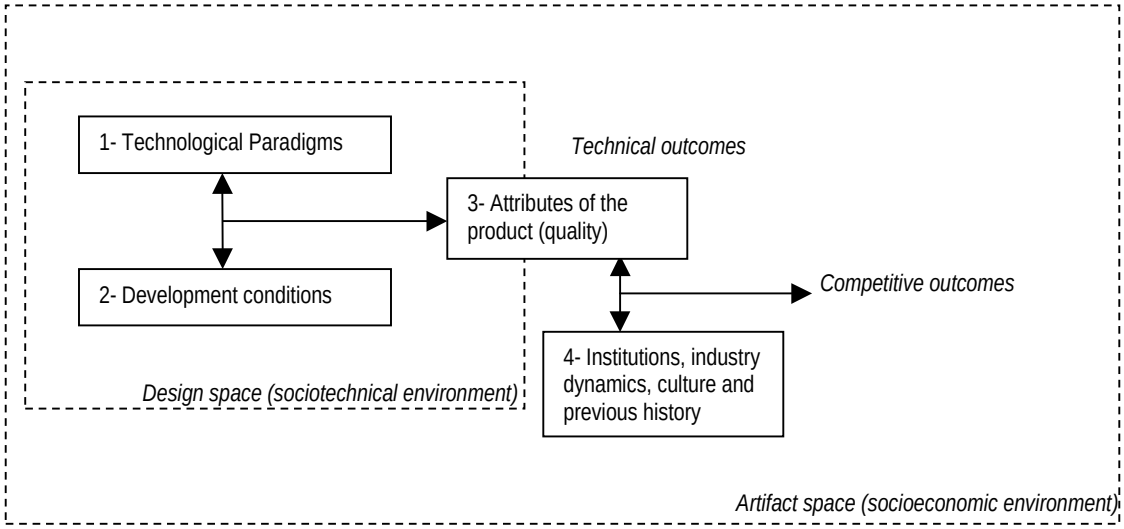
Therefore, we can consider that the design and development of an OSs is set in a certain environment that imposes constraints and limits possibilities for action. The interaction between that environment, described in section 2 and the development model followed, as defined by the technological paradigm, determined what I will call the attributes of the product, which could be considered as different dimensions of its quality. Those models better suited for the development of a product, given those technical constraints will be more efficient and, therefore, more useful for the creation of high quality products. When I speak about quality, I will to be referring to a "dynamic" concept. Since software products are being continuously improved, extended and connected into different systems, their attributes and characteristics should be analysed from an evolutionary perspective. These issues are placed inside section 3 and are based on literature about software and its desired characteristics.

Quality, however, is not the only issue that matters when studying competition in the OS market. Institutional, social and historical factors affect some dimensions of the quality of a product and its potential adoption. In the fourth section the ways in which all those aspects affect one particular aspect of a quality, compatibility, and influence the scale of the network externalities associated to a product, are analysed. In order to do so, literature concerned with the dynamics and competitive strategies present in the software industry is used.

Finally, in section 5, the channels through which the outcomes of different technical, business and legal strategies are fed back into the actors´ knowledge pools, in a learning process, are briefly considered.

A basic sketch of this model is presented in table 4.

TABLE 4- Different components of the theoretical model



Outcomes provide feedback, considered in section 5.

2.2- THE COMPONENTS OF THE MODEL

2.2 a) Technological paradigms

The concept

According to Giovanni Dosi, a Technological Paradigm can be defined as a "*pattern for solution of selected techno-economic problems based on highly selected principles derived from the natural sciences*"[21]. We can consider a technological paradigm as constituted by an exemplary (a product with a certain set of characteristics, a basis for development) and a development model (the heuristics followed in the evolution and incremental innovation of that exemplary). The development model, in the terms used previously, can be understood as a "design philosophy" or abstract pattern for handling problems that comprises the methodologies followed by the community inside the paradigm. As we will see in the case of software this has important organisational connotations. Technological paradigms define technological trajectories, *"the activity of technological progress along the economic and technological trade-offs defined by a paradigm"*[22].

Culture and ideology

Technological paradigms can be considered as "ways of defining technological problems and solving them" from a dynamic and evolutionary perspective. Embodied in the technological paradigm is the culture of its members, something that Dosi does not consider explicitly[23]. If we are to accept that technology is not something that exists outside people, but rather something developed *by* people, we should not forget their motivations, incentives and objectives. This aspect is specially important in the case that I will discuss, as some very important differences between the members of the competing paradigms can be observed and will be analysed in part 3, when studying Microsoft and Linux communities.

Organisational manifestations of the technological paradigm

In the case of software, different development models will be very strongly associated to different organisational arrangements (structures for the co-ordination and management of labour). This stems from the human-capital intensive nature of software development: software programming is an inherently creative task in which automation of relevant tasks is difficult to achieve and there is a strong reliance on individual workers. The ways in which labour is organised in order to carry forward the design, implementation and testing processes constitute the essence of a development model[24].

Different organisational arrangements can be considered, to a certain extent, as the result of the interaction between the objectives of the individuals involved in a productive activity, as determined by their culture, and the physical constraints posed by the nature of the activity. For example, we could consider different sets of objectives and different sets of strategies. The potential arrangements for the production of, let us say, cars, would be very different depending on the objectives of those responsible for that activity. The objectives determine constraints. For example, if a person wants to build cars in order to sell them and obtain a profit, the organisation devised will be very different from

[21] Dosi (1988) p. 224. See also Dosi (1984) ch. 2.

[22] Ibid. p. 225. My dissertation could be considered as an assessment of the future potential for improvement inside two different technological trajectories defined by competing paradigms representing Microsoft Corporation (Closed Source) and the Linux Community (Open Source).

[23] Thomas Kuhn, originator of the term, on the other hand, alludes to the "shared values" and "ideological" assumptions made by those inside a scientific paradigm. After all, another way of defining a paradigm would be as a "way of looking at the world", with metaphysical considerations included. See Kuhn (1996).

[24] We could say that the development model, in the case of software, has very important cooperation, communication and coordination aspects, that will be more thoroughly analysed in the next part of the dissertation.

one created by someone whose objective is to maximise the well-being of workers or the quality of the cars produced. The hierarchical firm could be considered, to a certain extent, as the result of a "maximisation of profit" aim. Given that objective, this organisational arrangement is quite efficient for the accomplishment of productive activities[25]. So we could say in less formal terms, that what an actor wants to accomplish (his objectives), determines to a big extent the ways in which he may act[26].

As we will see later, in the third part, when comparing Microsoft and Linux, the different objectives of different actors determine the constraints to which they are subject and the potential methods for development that they can adopt. For example, if the objective of someone is to maximise profits, the space of possible strategies is reduced for that person, when compared with somebody whose main concern is quality[27].

Of course, for most orthodox economists the previous discussion will seem pointless at the very least, since one of the basic assumptions in Neo-Classical economics is that maximisation of profit constitutes the main objective of the firm. On the other hand, when analysing the behaviour of Linux, we should challenge that assumption and accept the possibility of diverse motivations and incentives for individuals[28].

We find, therefore, that a technological paradigm, when considering the culture of those inside it as one of its constituent parts, can be associated to a certain development model, considered as an organisational arrangement through which a "technological trajectory unfolds". The result of this unfolding, that is, the efficiency of the paradigm in developing a product that satisfies its customer, will be influenced in a very important way by the environment, which will be analysed now.

2.2.b) The development environment

I have decided to detach these questions from the previous conceptual block, even if they are inextricably linked, in order to establish a clear border between what is internal and exclusive to each of the competing paradigms I will focus on, and those external conditions that both have to face. By doing so, it will be easier to distinguish between what people try to accomplish and why, and how the conditions of the environment in which they are set interacts with their efforts and strategies, finally determining their results. This section is mainly focused in the conditions of complexity in which software developers perform their tasks.

As Norman says, *"there are two faces to the complexity of a device, one internal to the machinery, the other external to the world and to the user"*[29]. When analysing the environment in which the developers of OSs work, I will focus on both faces. First I will analyse which internal characteristics of OSs make its design and development difficult, and then focus on what further problems are caused by complexity in the user and its needs and complexity in the systems of which the OS is just a component.

---

[25] It is not my objective to develop a theory of the firm and I will not discuss whether its objectives are maximise or find a satisfying amount of profits.

[26] His/her results will be determined, as we will se in section 2.2.b, by the suitability of the adopted strategy to the environment in which he/she is acting.

[27] Consider the first person as maximising his profits while subject to the constraint of maintaining a minimum level of quality (in order to satisfy its customers and be competitive in the marketplace), with somebody whose only concern is to maximise quality. The possibilities for that second person seem to be wider. I will consider all these issue later and apply them to the case of Microsoft and Linux.

[28] For some classical arguments in favour of the maximisation of profit thesis see Alchian and Allen (1964) and Friedman (1953). For a discussion of the methodological problems of this thesis, Blaug (1980). I have discussed this issues elsewhere (see Mateos (2001) pp. 6-10).

[29] Norman (1998) p.167.

To finish this section, issues having to do with the institutional (legal) framework, that can be considered as another dimension of the environment in which actors are set, will be mentioned.

Internal complexity

Something has already been said about the nature of software in the first part of the dissertation. Software programs can be understood as instructions that are followed by computers in order to perform tasks. These instructions are written in programming languages composing structures that the hardware can understand and process. It is the case with any language that mistakes (whether conceptual, grammatical, syntactical or orthographic) in the structures used to communicate can cause misunderstandings. This becomes especially relevant when we focus on computers, which need completely specified and understandable instructions in order to be able to perform their tasks correctly. Any kind of inconsistency in the code that consttitutes a program (a bug), will appear when the particular flawed routine is executed, causing crashes, errors and other problems[30].

This difficulty is made even more serious when we consider especially large software programs in which different sets of instructions interact with others, with other software programs (such as applications), hardware, input-output devices and networks. This is the case of Operating Systems, formed by millions of lines of code that should be coherently interconnected in order to avoid bugs. From this "network-like" character of software stems complexity, as a property directly related to the permutational increase in the number of potential connections, configurations and "states" of the system, each of them containing the possibility of a problematic bug or inconsistency[31]. The main strategy used to reduce uncertainty about the reliability of software products is that of testing before release. As it is shown later, however thorough it is, testing has many limitations and is unable to discover all the bugs hidden in software code. The result is the release of unreliable and "buggy" products.

Dynamic improvement

In addition to the static complexity of software, we should also consider the dynamic complexity that stems from its mutable nature. Brooks defines this "changeability" as another reason for the difficulties associated to software development. Since software is, after all, text (or "pure thought-stuff", in Brooks words), it may seem easy to improve it by adding a few instructions in certain routines, for example, or to correct bugs that have been localised by intervening in certain subsystems of the program[32]. This apparent flexibility, that makes the ever-continuing improvement and evolution of software programs seem possible, is however hindered by their intrinsic complexity. For example, as bugs are fixed, new inconsistencies (and therefore, bugs) are created. As new features are added, more complexity and potentially problematic interactions appear. This brings new difficulties to the improvement of the product both inside the firm (during the development process), and after it is released. Trying to solve problems will, in many cases, only bring more problems.

[30] Brooks (op. cit. n. 3) Ch. 16 classifies errors in two categories: essential, having to do with mistakes in the conceptual structure of software programs, and accidental, mainly syntactical errors such as the wrong spelling of instructions. According to him, while the number of errors in the second category can be reduced through certain tools, those inside the first one are much more difficult to avoid, since they stem from the essential characteristics of a software program: complexity, conformity, changeability and invisibility. In considering "internal complexity" of the computer I will dealing with the questions of complexity, changeability and invisibility as defined by him.

[31] For a graphic description of this problem see Sterling (1993).

[32] Compare the difficulties associated to the upgrade of a software program through the Internet with the improvement of a batch of already released cars or television sets.

Complexity in the management and co-ordination of software development

As I said before, software development, the task of writing code, is a human capital-intensive task, and some authors consider this as one of the reasons for the productivity problems that have been observed in the industry[33]. Human capital-intensive is a way of saying "labour-intensive", and in the mind of an economist, this will be associated with a small margin for productivity increases[34]

Automation of software development is so difficult because of the problems associated with the routinisation of the tasks being performed by programmers. After all, they are involved in an inherently intellectual, creative work in which continuous design, non-obvious, decisions should be taken. Computers are not able yet of handling that kind of work, and therefore can only be used as tools or assistants in the development process[35].

Given the labour-intensive nature of software development, the problem of managing it becomes, in many cases, a question of personnel management and organisation[36]. Since software, and specially OSs, are composed of so many interdependent parts, there is a need for teams to co-operate in development. The problem of co-ordination becomes very relevant in these circumstances, given the importance of an smooth integration of the interacting components of the system. Frederick Brooks, when speaking about his widely cited classic study of software development for IBM's System 360, the Mythical Man-Month, says that "*it is only incidentally about software but primarily about how people in teams make things"*[37]. I have already said before that the "development model", considered to be associated to each of the competing paradigms that will be analysed during my dissertation, have much to do with organisational arrangements followed in the design, improvement and enhancement of software products. Organisation, co-ordination and communication are essential subjects in the case of software production.

Another problem for software development associated to its human-capital intensive nature is the question of dependency on specialised personnel. The complex and to a certain extent "artistic" nature of code is associated to important tacit components in the skills of experienced developers[38]. Different developers tend to specialise in different subsystems in which they have been working and in many cases, the organisation may depend on them for their completion. A key, expert, individual leaving the project may become a catastrophe[39].

---

[33] "*Software development remains esentially a handicraft activity and is, so far a stagnant service*" (Baumol, Blackman and Wolf (1989) P. 136.)

[34] Almost since Adam Smith (1776), increases of productivity have been associated to the automation of productive processes. Smith was concerned, in his famous pin factory example, with the possible productivity gains to be derived from an specific organisational arrangement (division of labour). We could say that routinisation of tasks is a predecessor of automation, as its concern is to make the worker perform as a "machine". The own Smith says that specialisation provides incentives for innovation and therefore, automation.

[35] See Brooks (op. cit. note 3) Ch 16 and Cusumano and Selby (1995) p. 88 for some examples of automation as a tool for software development. An eventual paradox arises as we consider the automation of software development. When we speak about automation we are referring to the substitution of capital for labour. It seems obvious that that "capital" replacing human programmers would be computers. Those computers would need software to perform their tasks. Therefore, it would be necessary to develop software for the automation of software development and so *ad nauseam*.

[36] See for example Boehm and Ross (1989), in which all the management problems in software development are studied through an analysis of the incentives and motivations of diverse constituencies, inside which customers and users are included.

[37] Brooks (Op. Cit. note 3) p. 254

[38] Programming experience and skill is in many cases acquired through slow, gradual, "learning by doing" processes.

[39] "(software complexity) *creates the tremendous learning and understanding burden that makes personnel turnover a disaster*" (Brooks op. cit. note 3 P. 184)

External Complexity

Until now, the discussion has been related to the internal, intrinsic complexity of software programs as standalone constructions, and the problems associated to the co-ordination of workers engaged in their development. However, as Brooks points out, "*the software product is embedded in a cultural matrix of applications, users, laws, and machine vehicles*"[40]. When I speak about "external complexity" I am referring to the fact that software should conform to users needs and other components of bigger systems . The complexity of these other elements to which software must adapt adds new difficulties to its design and development.

User needs and complexity

The fact that software is designed in order to make computers useful for people means added complexity and difficulties to its design. We should take into account the fact that one of computers (and therefore software's) main tasks is to serve as tools for intellectual work and the co-ordination of collective activities. Designing them is a much more complex task than creating tools meant to help in the performance of physical work, as intellectual work is much more difficult to define[41]. Social, anthropological, psychological and linguistic issues should be considered in the creation of software if it is to adapt to the needs of the user and be truly useful[42]. As Operating Systems define basic graphical interfaces, the environment in which computer users operate, they are essential in determining the usability of computers and applications that depend on them.

There are problems in users defining what they want from their programs even when these are being specifically designed for them (in what is called "customised programs")[43]. The difficulties in building for the user grows in the case of general-purpose programs with "large user sets" (this is the case of Operating Systems). The needs of the user should be defined and provided for. This creates the temptation of "creeping Featuritis", that is, "*overloading the product with features of marginal utility, at the expense of performance and even of ease of use*"[44]. Trying to satisfy the needs of a not very well defined user introduces more complexity in the design of software.

One of the causes of the problems in defining user needs, even in the more abstract sense (in what refers, for example, to making software products usable at the simplest level) is the distance and differences between developers and potential users. As Boehm signals, "(programmers) *implicitly build up a set of assumptions about software users*[45]". Landauer has pointed out that software developers as a social group have different characteristics from the rest of the population and, therefore, it is difficult for them to understand the needs of potential users[46].

---

[40] Ibid p. 185

[41] I have not any notice of actual attempts to perform Taylorist time and motion studies for intellectual activities. Their "invisibility", as in the case of software, appears as an issue here.

[42] See Greenbaum and Kyng (eds.) (1991) for an account of issues having to do with the design of useful Information Technologies in a cognitive and organisational context. For some philosophical discussion on the design and development of computers, the nature of computer science as a scientific discipline and the relationship between humans and computers, Floyd, Zúllighoven, Budde and Keil-Slawik (1992).

[43] Brooks (op. cit. note 3 p. 199). Organisational complexity in many cases makes difficult a definition of what computers should do, and can generate further complexity in software. See also Winograd and Flores (1986) Ch. 12.

[44] Brooks (Op. cit. note 3) p. 258. See also Landauer (op. cit. note 5, p. 127): "*Almost all users use their computers for only a few operations. Nevertheless, their machines and minds are loaded up with a vast junk pile of options, commands, facilities, doodads, and buttons, most of them superfluous to the user and there just because somebody knew how to program it*".

[45] Boehm and Ross (Op. cit. note 36)

[46] Human-centred, human-involved design and usability testing constitute partial solutions to this problem. Some of them will be mentioned in the next part.

Therefore we find that there are many problems associated to the definition of users' needs and the way of providing for them through the specification and design of software products, and this brings increased difficulty to their development process.

The use of software

Even when user needs have been provided for, with more or less fortune, we find another problem stemming from the actual use that is made of the released products: we find the complexity associated to the variety of environments in which people put software products to use. When devising a testing strategy for software products before release, testers create what is called "user scenarios"[47]. They should imagine what users will try to do with their software and find out whether the execution of those tasks will cause problems due to hidden bugs. However exhaustive testers are, in any case, it will be impossible for them to do all the imaginable things that an user can try with a software program. There is a "*very large number of combinations of product usage scenarios that can occur. The various commands, data inputs, and underlying system configurations can cause a possibly infinite number of combinations*"[48]. Even after a product has been thoroughly tested, such in the case of Microsoft Windows releases (that are subject to extensive beta testing, that is, the test of the product before release by people outside the firm, who provide feedback about problems and bugs), the only real way of achieving true stability and reliability in a program is through its actual use in real conditions and its incremental improvement once underlying problems are discovered.

Software as part of a system

I previously quoted Brooks when he referred to the fact that software is just a component of a much larger matrix that also includes users, laws, applications and machine vehicles. Having considered those problems in the design and devellopment of software caused by the complexity of users, I will analyse the additional difficulties associated to the need of OSs being compatible and adapted to hardware and other software such as applications. I will focus on how that need for compatibility adds extra complexity to the activity of software development.

-Inter-operability

First I will consider the complexity of designing software as a component of a much larger system: there is a very large number of potential combinations of software, hardware, device drivers, applications and networks. It is impossible for software developers to test all these combinations looking for hidden inconsistencies and bugs in the code. The possibility of unexpected problems arising from unexpected combinations makes it impossible to assure the perfect reliability of a software program.

-Moore´s Law and its implications

In addition to the static complexity of designing software for its integration in bigger systems, there are the dynamic difficulties associated to the need for change in software in order to conform to the evolution of other parts of those systems.

The fast growth of hardware processing power defined by Moore´s Law sets the pace for a rapid evolution in the whole ICT sector, including the software industry[49]. In order to be able to adapt to the new, more powerful chips and machines and make an efficient use of their growing computational capacities, software needs to be continuously altered. As I said before, there are many problems

[47] See Cusumano and Selby (op. cit. note 35), pp 85-88 and pp. 294-316.
[48] Ibid p. 310.
[49] See Brown (op. cit. n. 2) for some implications of Moore´s Law.

associated to the improvement and alteration of software programs. Old, robust, tested systems have to be replaced with newer, unreliable ones, better suited for the exploitation of the ever-growing computing power available. Not only do the internal changes in software cause problems, but also the establishment of links and interfaces with new, unknown hardware equipment.

-Innovation and change in the IT industry

It is not only hardware that changes. Innovation in the IT sector pulls the laggard sectors towards more innovation and improvement in a network of fluctuations and change. Improvements in input/output devices, graphical capabilities and applications create a strong drive towards change and adaptation of other software, with all the associated problems caused by its complexity.

Innovation should not only be understood as a question of new, improved components of the system springing up in different sectors of the IT industry network, and driving forward other sectors. Broader tendencies such as the apparition of the Internet or the spreading use of networking, also create a necessity for changes in software. These great technological changes and organisational innovations are inherently unpredictable and, as IT industries have started moving in what has been denominated "Internet Time", demand an even greater speed response[50]. Important trade-offs between this required speed and the necessary carefulness in the development of reliable software programs appear.

TABLE 5- The environment in which operating Systems are developed. Summary of problematic factors.

Internal complexity: Software programs (and specially OSs) are complex constructions.

-Problems of consistency in a network of interconnected algorithms. Inconsistencies in the internal structure of subsystems or the external linkages between different subsystems produce bugs.

-Problems with the alteration of subsystems of the program caused by its interconnected nature.

-Problems in the coordination and management of software development.

-Problems with personnel turnover by the learning by doing nature of programming skills.

External complexity

-User complexity: software users are heterogeneous individuals with complex needs.

-Problems in defining the needs of the user.

-Problems in creating usable and useful artefacts.

-Problems caused by the very large number of user environments in which the product will be used. Potential apparition of previously unperceived interactions and inconsistencies.

-Systemic: software programs are just a component of much larger systems

-Problems caused by the very large number of potential combinations of hardware, applications and network environments in which the product will be used. Potential appearance of previously unperceived interactions and inconsistencies.

-Need for fast change and rapid adaptation to changes in other components of the system, specially hardware.

Complexity inside complexity inside complexity

Software subsystems form software systems that are just a component of computers. These computers are interconnected in networks and being used by complex users. If we add to this "systemic complexity" picture the necessity for fast change and adaptation and the fact that inside each possible interaction there may be a bug that will cause problems and crashes, we can start to visualise the kind of problems that software developers face. The analysis of actual practices for software development constitutes, in reality, a case of study of some methods for the potential management of complexity.

[50] For an analysis of the increased speed on the rate of innovation and improvement in products in the software industry see Cusumano and Yoffie (1998), specially Ch. 3 and 6.

The legal framework

The legal framework forms a part of the environment in which software developers (in a broad sense, including those who finance their activities) work. It does not interact with their development models in the same way as the previously discussed complexities of the technical and sociotechnical environment do, but it affects their incentives and can eventually determine whether the own development models are ever started[51].

I have spoken about the different motivations for software developers, simply referring to the fact that people do things, including the most technically complex, with a motivation, that can range from the search for the profit that can be obtained by selling that new technology, to the sheer enjoyment attained as a result of performing a creative work. When considering these motivations and analysing the legal framework, specially in what refers to property rights, we can conclude that it can affect the behaviour of individuals in a very important way by determining whether they can fulfil (or try to fulfil) their motivation by engaging in a determined activity[52].

The property rights framework, by decreeing whether the profits from an activity are individually appropriable or not, can influence to a great extent the incentives and behaviours of individuals. As we will see later, the intellectual property regime, by determining the appropriability of ideas and constructions such as software, influences the behaviour and potential strategies to be adopted by individuals engaged in software development, and therefore is a very relevant question for my dissertation.

2.2.c) Attributes of a software product and their determination

I have already spoken about technological paradigms as defining a development model (a methodology followed in the incremental improvement of a product) that is carried forward in the previously described technical environment of complexity. The suitability of that mentioned development model to the environment in which it takes place determines the results of a project, the attributes of the product. When I say attributes I am referring to a series of desired characteristics, dimensions in the quality of the product that will be defined and described in this section.

Software as pure design and development

Software production is different from many other industrial activities in the sense that its physical production is an almost negligible issue. It consists of the reproduction and packaging of a master copy, which is the result of the development process[53]. In most industrial activities, this does not happen. There is a design process of a prototype followed by the mass production of the product that will be sold in the market. There is a separation between the design laboratory and the factory[54]. Both spaces are important, and there is a difference between the creation of the product (analysed as Research and Development) and its production (analysed as a process, although it will also have development aspects).

[51] The theoretical framework underlying this discussion is the "rules of the game" theory (see Baumol (1992) and and New Institutional Economics analysis of property rights as exposed in, for example, Eggertsson (1995).

[52] While a car producer motivated by profit would stop building cars if the legal framework established that it is illegal to sell cars with an economic gain (he would produce something else or even engage in criminal activities), someone who just built cars because he enjoyed doing so would not necessarily.

[53] With the introduction of the Internet, even packaging has disappeared as many software products may be directly downloaded into users computers.

[54] Of course this delimitation is not clear-cut, as the design lab obtains feedback from the factory, modifies the design etc.

In the case of software, design and development are the main production process, and determine the final characteristics of the product. As Paulk says "*The basic premise underlying* (our work)…*is that the quality of a software product is largely determined by the quality of the software development and maintenance processes used to build it*"[55]. Quality appears as the result of an efficient development process, and therefore, when trying to determine an organisation's capabilities for the creation of high quality software products, we should focus on the strategies and methodologies that are being used, and constitute that process.

The dynamic aspect of quality

It is very difficult to consider a software product as ever finished. Its nature is changeable and evolves continuously. Software is altered in order to adapt to change in other components of the system, to solve problems after they are discovered, or to improve its performance. This process of incremental improvement, circular redesign, redevelopment and testing of the mutable product takes place at a much faster rate than in other industrial products. Therefore, we should not analyse the characteristics of software as the more or less final, static result of a long development process (there is no such end to the development process. at least in complex systems)[56]. Instead, the analysis of software attributes should be made from a dynamic perspective. When I define quality I will be referring to the potential speed at which desired attributes can be reached *before the next change*[57].

Quality in a software product: The attributes.

I will consider performance, reliability, functionality, manageability, compatibility and innovative potential as the main dimensions of a software product's quality. The previous categories are quite loose and maybe not equivalent to the ones used in the software industry. In any case, I expect to cover most of the important aspects of Operating Systems in the following comprehensive definitions.

-Performance: When I say performance I am referring to the OS´s capacity in making use of the computing power of the hardware it is connected to. That is, its speed in performing tasks as compared to its memory requirements. Performance in an issue determined to a big extent by the design of the OS and the quality of its code.

-Reliability: By reliability (also called stability or robustness), I mean the extent to which an OS is free of bugs and errors that can cause crashes and other problems. Previously I referred to these issues, and showed that the unreliability of software is mainly caused by its essential complexity. The way in which different development models deal with this complexity is very important in order to determine whether acceptable reliability is attained.

-Functionality: Functionality can be understood as the breadth and depth of the functions and tasks that the product can perform. Of course, functionality is related to performance and reliability by the fact that new features mean added complexity that may affect the previous attributes[58].

---

[55] Paulk (1995) p. 3

[56] For example, as soon as a new product is released, Microsoft developers start working on its update, in what is called Milestone 0 (see Cusumano and Selby op. cit. note 35 pp 206-207).

[57] Consumers of software do not (and cannot) expect a perfect product, but one that will be supported, improved and made more reliable in the future. That is why Eric Raymond, when analysing software production in his essay The Magic Cauldron, considers software development not as a manufacturing activity but as a *service*: the client is paying the software company for maintenance in the software service that he has bought. See Raymond (2001). For an on-line version of the Magic Cauldron, see http://www.tuxedo.org/~esr/writings/magic-cauldron/ accessed 16/7/2001.

[58] Inside this attribute I will also include security, an issue that in many cases is deemed to be so important that it is assessed in its own category. OSs, by establishing the connection of different computers in networks, are very important in determining the security policy and procedures that make sure that parts of the system are not accessed by unauthorised users.

-Manageability: I will consider manageability of an OS from two perspectives:

-first, from the perspective of the individual user, I am referring to the usability of the system and the difficulty in learning how to use it.

-In a broader way, I am also alluding to the manageability of the OS from an administrators view. That is, its possibilities for reconfiguration and customisation in order to adapt to the needs of an organisation.

Both these issues are influenced by the development model, since it determines the kind of interaction between programmers and future users.

-Compatibility: The issue of compatibility is very importantly linked to the concept of network externalities mentioned in the first part of the dissertation. We could define compatibility in a very broad way, by saying that it is *"the capacity of an OS to make use of the productive resources available in the network*". Those resources include the possibilities to share information with other computers, to make use of applications and to be connected ("ported") to different pieces of hardware. As we will see in the next section, this issue is specially affected by the dynamics, structure and competitive strategies that appear in ICT industries.

-Innovative potential: Innovative potential is an abstract issue that has to do with the possibilities of innovating in a determined OS, that is, introducing new features, architectures and arrangements that increase any of the previously mentioned attributes. Since the development model determines how is the product designed and implemented, it also influences the way in which individuals can innovate and incorporate new ideas in the OS.

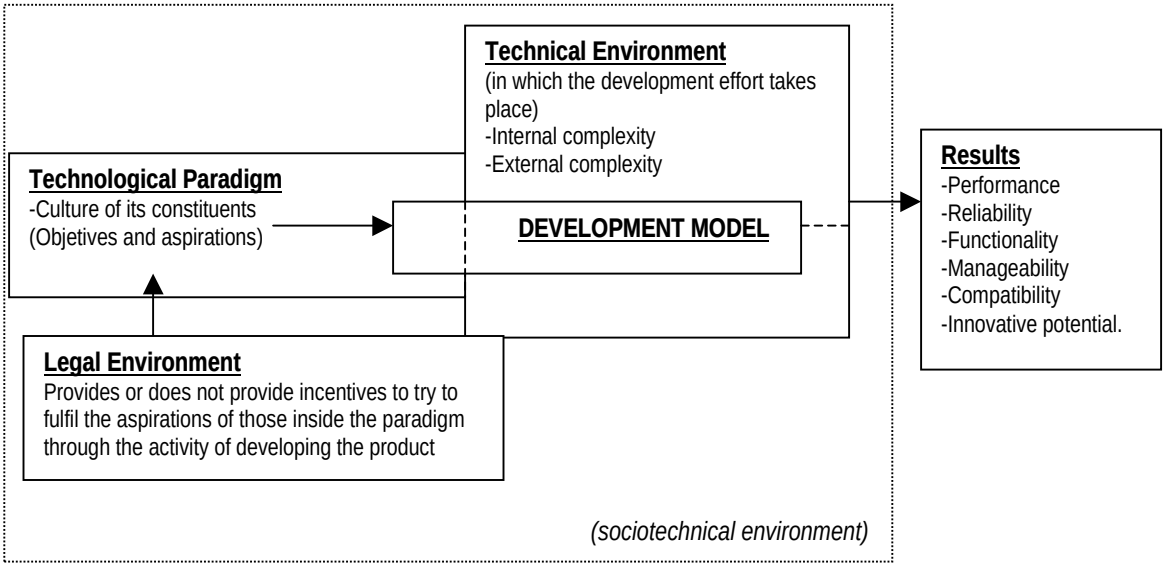
A summary so far

I have defined technological paradigms as techno-cultural categories that embody the objectives and aspirations of their constituents. A technological paradigm also comprises a development model, which is the methodology used by the members of the community inside the paradigm to improve and enhance their product, and that in the case of software development is very strongly associated to the organisational arrangements adopted by the community of developers[59].

Development takes place in a certain environment, composed by technical and legal factors. The results (attainment of certain desired attributes of the product) are determined by the suitability of the paradigm to that considered environment, by what we could consider its "fitness". Those development models efficient in dealing with the essential aspects of software (mainly complexity, as considered before), will be more successful in the dynamic improvement and adaptation of their product to the needs of their users. The legal environment, on the other hand, may influence the incentives of the actors and either impulse or deter their motivation to engage in the productive activity. All this is summarised in table 6.

[59] The "their" is an important question. Products are after all, the definers of the existence of the technological paradigm and its community. We could consider the community of developers as a group of individuals clustered around that product. The apparition of technology as a agglutinative factor is interesting and could be examined from an Actor-Network perspective. See

TABLE 6- A summary of the first part of the model.



2.2. d) Competitive dynamics, structures and strategies in ICT industries

Until now I have been discussing, in an abstract fashion, the development of software products inside a technological paradigm and an environment which determines certain results, the attributes of the product, understood as different dimensions of its quality. I have not mentioned costs, prices or competition. But these issues are very relevant indeed. As Lecht says, "*the computer industry may be thought of as driven by economics and politics- to the dismay of technologists*"[60]. Competition in the marketplace determines the eventual success or failure of a product, and provides its developers with feedback about the efficiency of their strategies. And, as we will see in this section, quality of the product is not at all the only relevant factor in this market. Previous history, having determined industrial structures and user bases, and certain business strategies, specially in what refers to compatibility issues, can affect the adoption or demise of a product, irrespectively of its *a priori* technical quality. The way in which a product is introduced into the system and the question of whether it is able to connect with its other components or not is almost as important as its quality (as defined by its development model) in determining its usefulness and eventual adoption or neglect.

In the following section I will be studying some important economic and strategic factors related to this that characterise competition in ICT, and specially, software, industries.

Another part of the environment

The separation between this component of the model and the previous one is quite arbitrary. After all, economic and institutional structures form part of the environment in which a technological trajectory unfolds. However, I have decided to consider the following issues separately in order to make the analysis clearer. All the factors dealt with in the previous section can be considered, to a certain extent, as exogenous in relationship to the actors´ efforts (they constitute the immutable

[60] Lecht (1977) p. 1

environment in which they operate)[61]. On the other hand, the issues that I will be considering now are related to historical and institutional aspects of the software industry, which have been shaped by the decisions of actors in the past. In the same way, decisions that are being made now will affect the future environment and the possibilities of the competing paradigms.

Pricing

Before considering compatibility questions I will refer to the issue of pricing, that given the special characteristics of software products and their distribution channels, becomes a strategic variable that can be used to increase the competitiveness of a product[62]. Since most of the software production costs are incurred in during its development, once the "master copy" is created, the marginal cost of additional units of the product (especially in the case of distribution through the Internet) becomes almost negligible[63]. The pricing decision will not be made in marginal, neo-classical terms. Middle term predictions and dynamics are very important in determining potential pricing strategies.

Compatibility and network dynamics

In the following pages I am going to speak about some special aspects of competition in the software market. My aim is to show how although the quality of a product is, to a very big extent, determined by the development model followed in its creation, broader dynamics having to do with previous history and prevailing industrial structures can affect the possibilities of success of different products. I will try to analyse how they can affect one particular characteristic of a product's quality, compatibility, making it either more or less desirable for consumers, and influence the network dynamics associated to the diffusion of a product.

Compatibility was defined before in quite a broad way. It encompasses the capabilities of an OS to operate in different machines, to make use of applications and to communicate with other computers. The compatibility attribute measures the adaptability of OSs to a bigger system of which they are just a component. Therefore, it becomes a vital issue in determining whether a product will be adopted or not. An incompatible product, and specially an OS, becomes useless in a networked environment.

1)-Standards and path-dependencies: Standards in software industries are deeply related to the network dynamics mentioned in part 1 The need for compatibility creates positive feedback: once a technology is adopted, network effects that propel that technology towards domination of the market, with its eventual establishing as the *de facto* standard, appear[64]. The establishment of standards in the past creates "path dependencies": With this term I refer to the fact that past events and decisions limit the scope for present ones. When we speak about technology, we could consider that once an investment is made, the space of future possibilities is constrained. Sunk costs, whether in capital or training can discourage users of a certain technology from change to another one, even if it is

[61] Although the previously mentioned aspects of complexity in the development of a software product can be attenuated through the use of certain tools, we can consider it as "given" or essential, in Brooks' words. On the other hand the institutional and legal framework can be altered, but not directly by the actors but by political forces. Of course the existence of industrial lobbies affecting the legal framework should be acknowledged. Actors can influence the legal environment, but in a less straight fashion than the variables that will be considered in this section. More will be said about the strategic use of the legal environment in part 4.

[62] We could speak about the "value for price" variable considered by many software reviewers. In that sense it would constitute the final attribute of quality as considered previously.

[63] See Shapiro and Varian (op. cit. note 7), ch. 9 and Cusumano and Yoffie (op. cit. note 50) Ch. 3. This aspect of software lies at the heart of the question about whether "information wants to be free".

[64] As I have mentioned, this kind of network dynamics are, in the case of software, mainly caused by the need for compatibility (homogeneity in the communication standards) between different components of the system.

superior, in what has been defined as "technology lock-in"[65]. We could consider this concept in the following terms: imagine two technologies A and B, of which B is more efficient. If A and B had been released simultaneously, B would have captured the market. On the other hand, if A has been released before, with no competition, it will have been adopted and therefore, captured the market. A technology entrenchment phenomenon may have taken place, since users have made an investment in technology A which makes it difficult for them to switch to technology B even if it is superior.

2)- Technology lock-in and network dynamics Network dynamics create collective switching costs, another barrier to new entrants in the market that reinforces the entrenchment of technologies: imagine a market with a hundred users that communicate with machines of a technology "A". Now a different technology, "B", is introduced into the market. B is more efficient than A but it is not compatible with A Although all the users would be better off if they simultaneously switched to standard B, no individual agent has incentives to follow this strategy. After all, if he substituted B for A, he would lose the capability to communicate with the rest of the users[66].

A potential strategy for the users of technology B would be to make it compatible with A, in what has been called the "*creation of a migration path for existing users*[67]*".* However this kind of behaviour presents potential problems, both technical (the achievement of compatibility may be difficult if the differences between the competing technologies are important) and legal (there may exist legal obstacles to the reproduction of communication standards in order to achieve compatibility).

3)-The importance of industrial structures and institutional arrangements: I have already mentioned the fact that, given the interconnected nature of ICT technologies, a certain degree of co-operation between different firms is necessary for the creation of coherent, functioning systems[68]. Collaborative networks and consortia span across different sectors in ICT industries with the objective of sharing information and managing collectively technological developments and their integration through the creation of communication and inter-connection standards. However, the existence of these networks may become an obstacle for potential entrants. Explicit and implicit arrangements between the incumbent actors may make access to the market very difficult for potential competitors[69].

In the case of OSs, incumbent players can use their influence and industrial contacts in order to deny competitors the access to vital resources such as the support from hardware producers and vendors, applications developers, and distributors, or ostracise them from the industrial standard-setting processes[70]. Given the previous definition of compatibility, we could understand these industrial arrangements as affecting the quality of a product by, for example, reducing or nullifying its compatibility with hardware or applications. We could speak about a potential entrenchment of incumbent players, associated to certain products and technologies, impeding access of potential

---

[65] See David (1975), David (1986) (also available as a web document at http://www.stanford.edu/group/mmdd/SiliconValley/David/QWERTY.html Accessed 15/8/2001) and Arthur (1992) for a presentation of these issues. For an applied analysis to the case of software, see Shapiro and Varian (op. cit. Note 7) Chapters5 and 6.

[66] Only if more than fifty users switched at the same time to technology B, would the rest of them have incentives to do the same Now compare the described setting with the "real" world, with millions of uncoordinated users. Also take into account the difficulties inherent to the task of determining the relative efficiencies of different technologies (specially important in the case of complex products such as OSs). The barrier established by the need for compatibility with dominant standards becomes an almost insuperable barrier for potential entrants. See David (1995) P. 25.

[67] Shapiro and Varian (op. cit. note 7) 191-195.

[68] Ibid. pp. 242-255, on the issue of "building alliances" and Pavitt (2000) (wd) (pp. 6-7) on the growing importance of this kind of collaborations, their reasons and some of their limitations. Mansell and Steinmueller (op. cit. note 12) Ch. 6 also show the importance of consortia in the integration of large ICT systems).

[69] The limits between inter-firm collaboration and collusion are in most occasions very thin.

[70] In this sense, they would be affecting the kind of inter-market network effects depicted on table 2.

competitors to the market. One possible strategy for potential entrants to this market would be to create their own industrial networks and partnerships.

4)-Expectations and credibility: Given the importance of continuous enhancement, support and maintenance for software products, a developer group's credibility may become another determinant issue in the dynamics of the market. When choosing between competing products, consumers do not just consider the "static" efficiency (as value for price) of a system, but also the perspectives for its future improvements and evolution[71]. Paraphrasing Joan Robinson, we could say that there is only one thing worse than being locked-in to a monopolist supplier, and it is being locked in to a moribund supplier. In many cases, mistrust and uncertainty about the future survival of a supplier will cause important obstacles for the adoption of a system, in what could be considered as another kind of switching costs[72].

Even if a potential entrant in a market achieves compatibility of its product with the dominant technology, the consumers may still feel concerns about the degree of that compatibility.. Trust is a very important factor in ICT markets, and uncertainty about new, previously unknown entrants, may deter users from switching to the new technology. Products with this kind of credibility problems are, in the software industry terms, defined as suffering of FUD (Fear, Uncertainty and Doubt) [73].

Credibility does not only affect consumers´ decisions, but also those made by actors in the ICT industry. As I have mentioned in the previous discussion, the creation of partnerships and arrangements with important players in other sectors is vital for the eventual usefulness and acceptance of a product. Trust and credibility issues become as important in this environment as in the end-users market (players in the industry may be reluctant to strike deals with "potential losers").

5) Trust and culture: credibility, as I have said, has much to do with trust, and trust has very important cultural aspects (it is not only related to economic incentives, which is the main perspective adopted during the previous discussion). Individuals tend to trust more easily those who have views similar to them and to distrust those that are perceived as "different". As we will see later, this has very important implications in the case of competition between Linux and Microsoft[74].

6)-Policy in networked markets: The economic incentives for turning a component of the ICT system into a standard are very important: its ownership can provide their owners with extraordinary benefits. In markets where network dynamics are present all the previously discussed issues become important in the determination of the eventual winner. As I have mentioned before, the dynamics of ICT markets are not only influenced by technical factors, but also by economic and political ones. Credibility, marketing and industrial arrangements can determine whether the network dynamics that characterise these as *tippy* markets get started, conditioning therefore eventual dominance by a certain technology and its subsequent entrenchment[75]. Although we should acknowledge this as a

[71] This creates some limitations for the strategic use of pricing previously mentioned. Price at a moment in time is just one of the factors considered by consumers.

[72] We can find cases of self-fulfilled expectations in ICT markets. That is, products "expected to succeed" just succeed. In the same way, products "expected to remain dominant" remain dominant (users may fear to switch to potential losers- lose compatibility with bigger networks as described in (2)). This is where the worlds of credibility, public image and marketing become intertwined.

[73] Incumbent players may use that kind of fear strategically by, for example, making statements about "perfect compatibility not being assured for users of the new system". For some examples of the use of this kind of "management of expectatives" tactic see Wallace (1997) and Goldman Rohm (1998).

[74] We could consider an analysis of the adoption of ICT technologies using constructivist concepts such as that of or alignment (Callon (1992)).

[75] See Shapiro and Varian (op. cit. note 7), Chapters 8 and 9 and David (op. cit. note 66) pp. 24-26.

reality, the fact that adoption of technologies is a negotiable issue opens the door for public intervention[76].

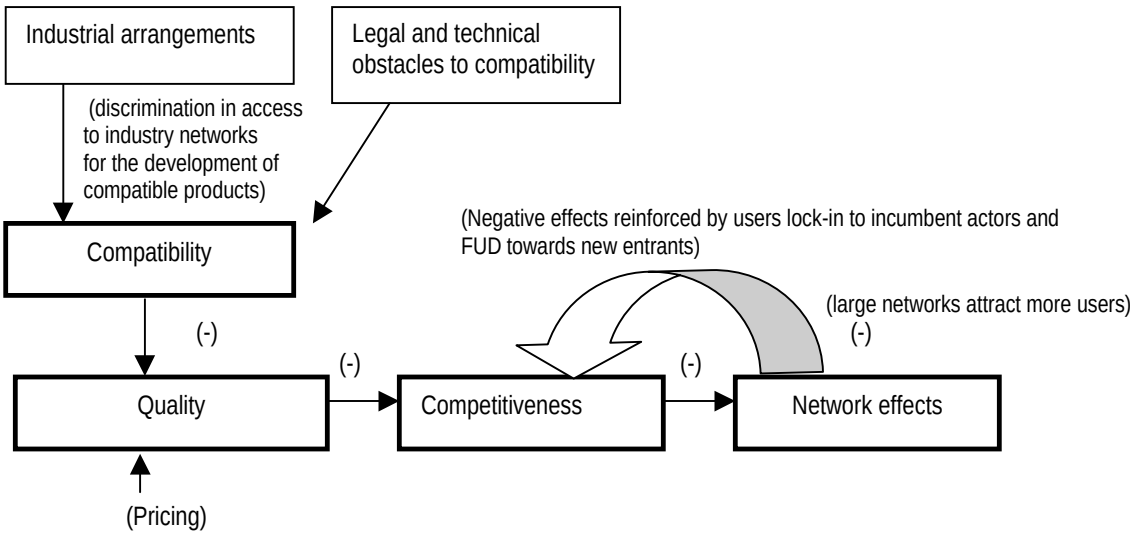
Open standards are an alternative in markets where network dynamics are present. Instead of having different groups competing with non-compatible technological options, a standard may be agreed by all the actors or by public standard-setting institutions, with competition taking place inside those arranged technical parameters that guarantee inter-connectivity. Open standards diminish the possibility for technology lock-in and therefore benefit consumers. On the other hand, the profits to be obtained by competing inside an agreed standard are smaller than those derived from owning a proprietary standard. Firms, even when apparently agreeing an open standard may have incentives to try to gain market share by fomenting incompatibility and capturing users in the practice that has been denominated as "*polluting standards*"[77].

7)- Summary: industry dynamics and quality of the product: In the theoretical model being developed I will consider the previously discussed issues as affecting the compatibility and network externalities of a given OS and therefore determining whether it will be adopted or not. I will assume that consumers select the highest quality product from all those available in the market, considering the various dimensions (attributes) of quality defined in section 2.2.c. However, compatibility as an attribute is not only determined by the development model followed in the construction of the OS, but by definition, also by the properties of the rest of the system with which this component should be compatible. There are structures determined historically (such as the dominance of another product with the associated technology lock-in, or industrial arrangements that discriminate in favour of incumbent players) and strategies (such as the management of expectations, credibility and compatibility) that can raise obstacles for the entrance of new competitors into the market. By making the insertion of new technologies in the existing sociotechnical environment difficult, their adoption will be impaired, and the kind of network dynamics that I have referred to will not start, condemning this product to a marginal use, if not to utter disappearance. A scheme of this discussion is shown in table 7.

[76] In a "technologically determined" world, the best technical solutions would eventually be adopted. In the real one, where negotiation between powerful actors can influence to a very big extent the future evolution of markets and information infrastructures, not always in the general interest, the public sector becomes another player that should be concerned with the eventual benefits for its members, that is, the citizenship. After all, uncertainty (imperfect information), externalities and market power, three "market failures", could be considered to be clearly present in the analysed sectors. For a discussion of these issues see Brooks (1986) and the analysis of technology performed by the "Constructive Technology Assessment" school (Schot and Rip (1996)). As Mansell and Steinmueller (op. cit. note 12) assert, the liberal discourse of "government hands off high-technology markets" is not bereft of values and is in many occasions used by incumbent players to defend their own interests.

[77] See Shapiro and Varian (op. cit. note 7) ch. 8.

TABLE 6- A summary of how ICT industrial dynamics affect competitiveness and adoption of new products.



2.2. e) Closing the model: feedback and learning

Possibilities for action and different actors

In the theoretical model there are two possible explicit channels and an implicit one through which actors can put different strategies into practice. The two explicit ones are first, the decision about strategies adopted for the technical development of the product (the development model) and the second one is the use of business strategies that increase the competitiveness of their product (or decreases their competitors´).

The availability of strategies of the second kind depends to a great extent on the industrial structure and power of the considered players. Usually, incumbent actors will have a greater menu of potential business strategies available, because of their bigger influence in the dynamics of the industry, already established user bases and credibility etc. On the other hand, potential entrants can use hybrid business-legal strategies such as anti-trust lawsuits in order to try to shift the power balances in markets by weakening existing structures or forcing compatibility of existing networks with their own products.

The implicit place where actors can act is the legal framework mentioned in section (b). I have already said that actors can try to alter it, for example through lobbying, it in order to increase their potential strength or diminish their competitors´[78].

Channels for learning[79]

Having considered the ways in which industrial structures historically constituted and certain dynamics, strategies and processes present in ICT markets affect the quality of a product, and therefore, its competitiveness and market results, the theoretical model will be closed with an allusion to the fact that actors do use the results of the development process and the evolution of the market in order to learn and improve the efficiency of their strategies, given their objectives.

[78] The previously mentioned anti-trust lawsuits would not fall into this category because in most occasions they are trying not to alter the legal framework, but to use the existing one for competitive and business reasons.
[79] This summary of learning modes and loci is based on Pavitt (1991) p. 46-47

The loci where important feedback about the suitability of strategies to the environment can be found are mainly three:

-First, the difficulties faced during the development of the product: As products are developed, problems and trade-offs are encountered. They constitute valuable knowledge that can be used to improve the development methodologies and heuristics followed by the community. This kind of learning would fall into what, inside evolutionary economics, is defined as "*Learning by doing*".

-Second, the quality of the released (and used) product: When the product is confronted with real scenarios, the community of developers can obtain important knowledge about its quality (and therefore, about the efficiency of their development model). Development strategies can be altered and refined in order to improve the quality of the product in subsequent releases. This kind of learning could be considered as "*learning by failing*" and "*learning by interacting"*.

-Third, the market results: The final and determinant judge of all firms who try to sell or have a product adopted by consumers is the market. The results obtained by their product in the competitive environment will provide the community behind the product with knowledge about the suitability of their technical and business strategies, in what we could call "*learning by competing*".

Cultural constraints to change

I started speaking about technological paradigms and the culture of its constituents, and I will finish mentioning this issue once again. I have said that feedback and learning may bring changes to actors´ strategies. However, we should not forget that these changes take place in order to increase the success of the community in achieving its objectives, as determined by what I have defined as its "culture". The values and aspirations embedded in the considered culture can make it sometimes difficult to change strategies and methodologies, with the ensuing conflict and necessary trade-offs between means and ends[80].

Channelling technological trajectories: constraints as focusing devices

I said before that this dissertation could be understood as an analysis of the relative efficiencies of two technological trajectories associated with different technological paradigms used in the development of Operating Systems. I have also spoken about constraints to the kind of decisions that may be taken by actors. They are cultural, environmental, legal and socio-economic, and limit the kind of directions in which technological improvement can take place . These "focusing devices" following Rosenberg's definition, condition in different ways the loci in which the "search" performed by a technological paradigm may take place and the methodologies put into practice[81]. In the next part I will consider some of the ways in which they affect Microsoft and Linux possibilities for technological improvement.

[80] Following the examples of the profit-maximising automobile maker: maybe he is not faring well in the market because his workers (let us assume not very qualified and badly paid in order to "maximize profits") are not concerned with the quality of the product (which therefore is worse than other available models). Maybe a change in his development model is necessary, but a conflict between efficient (in a competitive sense) production and maximising profits may appear. After all, we should not forget the incentives of the actor for starting the production of automobiles in the first place. I will now please ask the reader not to take too seriously the simplistic examples I have been employing. They are not meant as an attack against the profit-maximising behaviour or any thing similar. I just want to draw extremes with a didactic objective, as David Ricardo used to say he did, and criticise the mainstream assumption of "profit-maximisation equals productive efficiency".

[81] Rosenberg (1976) Ch. 6

2.3- CONCLUSION

2.3. a) Some methodological issues

The theoretical model developed during this part should be understood not as a tool for prediction but as a description of some of the interactions that take place between the micro behaviour of actors and the macro environment. Some of its components are quite general (such as the concept of technological paradigm) while others are applicable especially to ICT industries (the fourth component, dealing with competitive strategies, structures and dynamics). The one dealing with external complexity is very specifically related to some of the technical and organisational issues having to do with the development of software. In any case, the scheme can be useful, in my opinion, to analyse other sectors and industries, specially those engaged in the production of complex products in which the development methodologies followed and the interaction with users are specially important, and those in which compatibility and communication aspects, very deeply linked to network dynamics, are present[82].

In any case, the kind of framework created could be considered to fall inside what we call a conventionalist methodology (concerned with the classification and explanation of empirical processes). Given the complexity of the issues analysed and the breadth of the model, prediction appears as a difficult objective. The closest thing to a test of the model will take part in the following parts, where I will analyse Linux and Microsoft development strategies and their interaction with the technical, legal and economic environment using the preciously built framework. There, the reader will have to decide if this framework is useful for research, explanation and, if not prediction, at least consideration of possible future scenarios in the analysed industry. Application of this model to other sectors and products would be necessary in order to determine its generality.

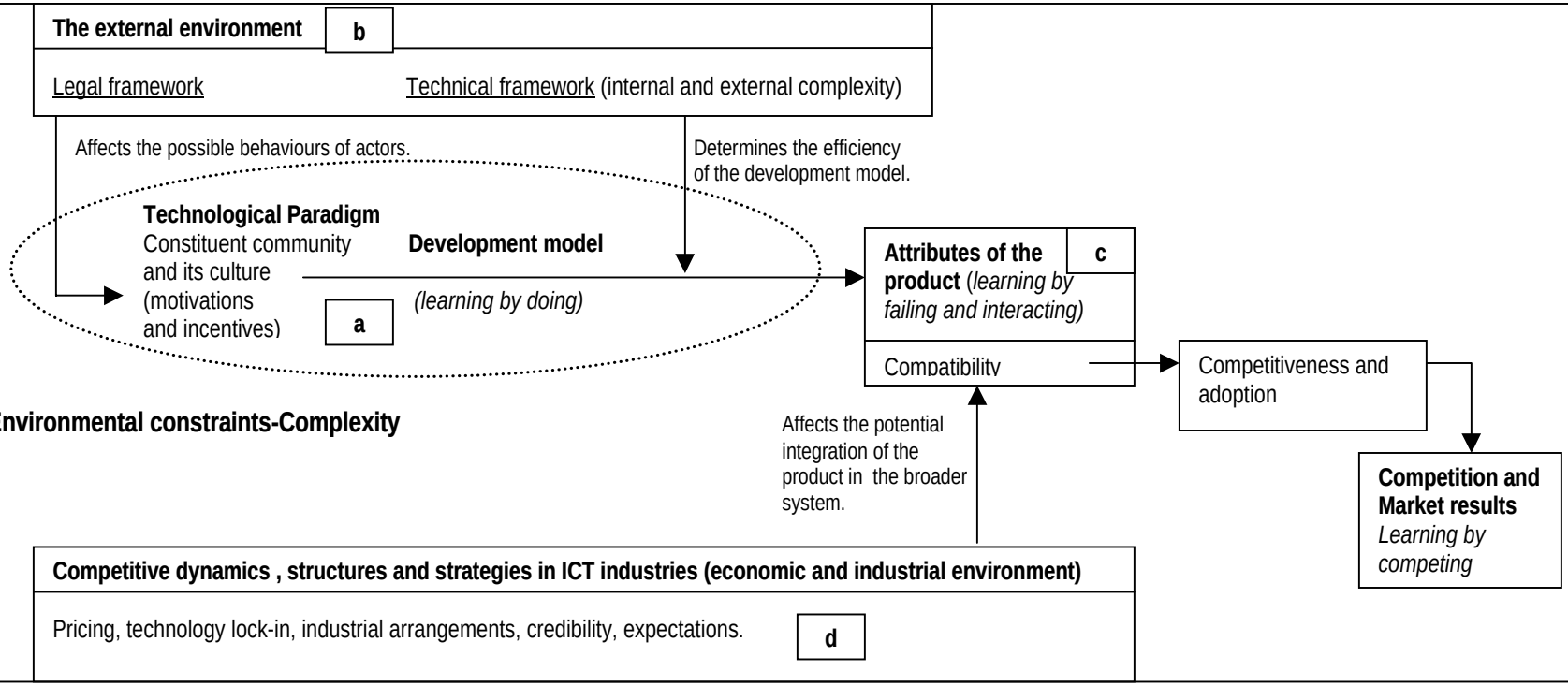
2.3 b) Last considerations

The model should be considered as a simplification of reality, and understood in an iterative way[83]. Although the classification of phenomena, concepts, problems and processes seems to follow a linear development, as the reader will see in table 8, it should be kept in mind that reality is much more complex than that. We are considering a whole array of actors working in different locations and activities and obtaining diverse kinds of feedback from the fast-changing environment. From this perspective, the model can be understood as the systematic analysis of a portion of the world, which I have tried to avoid distorting too much by being abstract and open.

[82] Most knowledge-based sectors share with software the importance of design and development, with large initial investments necessary but incremental production relatively cheap. See Brian Arthur´s quote in Battram (1998) p. 161

[83] The now almost "classical" metaphor of learning, change and innovation as adaptations to the environment (constituted by all the influential actors and structures except the analysed one) suits quite well the kind of history that I have been telling so far.

TABLE 8- The theoretical model



The boxed letters show the section of the model dedicated to the discussion of each of the "modules".
Italics show where is feedback for learning mainly produced, going back to the constituent community (this is analysed in section (e))

3-THE CASE STUDY

After setting the theoretical framework in the previous part, the empirical aspects of the dissertation will be presented now. It consists of an analysis of the development models used in the creation of Operating Systems by two competing technological paradigms, Linux and Microsoft. I want to assess their strengths and weaknesses, mainly from an organisational perspective. My objective is to analyse the relative efficiencies of the considered methodologies in dealing and managing the complexity that was described in the previous part and constitutes an inherent aspect of software development. Therefore, I am going to be alluding to the questions described in sections (a), (b) and (c) of the theoretical model, as well as to learning and adaptation issues that have been mentioned in section (e). Economic and business dynamics considered in section (d) will be left for Part 4.

3.1- THE METHOD

In the theoretical model I mentioned the importance of the development model in determining the attributes of the product, that is, its quality. The efficiency of the model in producing an acceptable, competitive product, depends, as I said in section (b), on its suitability to the environment. In this part, after a brief discussion of the characteristics and cultures of the different technological paradigms being studied I will analyse how do their respective development models deal with the complexity associated to the creation of OSs, and give some reasoned hypotheses about which of them could be expected to be more successful in this activity. Then I will consider evidence about the past evolution of these models and try to show how does it support the proposed hypothesis.

3.1.a) The empirical evidence

As I have said, in this part I will describe the different methodologies used by Microsoft and Linux in the development of their products with the objective of reaching some working hypotheses about their efficiency. Measurements of development activity rates and results, market evolution as indicative of the quality of the products or assessments of final products´ quality could be used as evidence with the aim of contrasting the proposed hypothesis. However, there are strong problems associated to the use of these different pieces of evidence:

-Although there is some quantitative data on the performance of Microsoft, such as its development times, bug detection rates or surveys about user satisfaction, we cannot say the same about Linux[84]. Given the necessity of comparability between both communities, the use of these available quantitative indicators will have to be relegated to a minor role.

-Market data will not be very useful as an indicator of the relative efficiency of the competing products either, given the issues that I have mentioned in section (c) of the previous part[85].

-Employing quality (as, for example, defined by specialised analysis of products by consultants or technical magazines) as evidence about the efficiency of a development model is also problematic. First, this kind of evidence tends to be quite controversial[86]. Another difficulty is that this quality is usually defined in a static sense, while my main concern is with the dynamic efficiency of development models.

---

[84] See Cusumano and Selby (op. cit. note 35) and Cusumano and Yoffie (op. cit note 50). The unavailability of data about Linux stems from the own nature of its development model, as the reader will see later.

[85] This is because the "technical quality of the product as determined by its development model" is a variable which can be considered as different from its "competitiveness", affected by business, strategic and compatibility factors.

[86] See for example Moody (op. cit. note 10) Ch. 16

Given these difficulties and problems, the main empirical data I will be relying on during the following case study will be qualitative, related to the development practices, organisational arrangements and their evolution in the analysed groups. What I will do is rely on descriptions about the methodologies followed by each of the competing technological paradigms in the development of their products. By examining some of the problems encountered in these processes , as accepted and discussed by the own developers working in the analysed groups, and the solutions, as well as learning processes and "best practices" devised to deal with them, I will show how Linux and Microsoft have tried and succeeded (or failed) to manage the inherent complexity of OSs. I will focus on the evolution of different development models and try to signal the tendencies hinted by those processes as evidence about learning and adaptation. These tendencies may show us, when examined under the light of the theoretical framework previously devised, which practices and methodologies are more efficient in the development of software and therefore provide or not support for the hypotheses proposed.

I acknowledge that the reader may find this kind of research too abstract, in the sense that numbers are usually considered to be the main paintbrush available for drawing reality, but given the relative youth of research on Open Source and its methods, I think that the kind of analysis that will be performed in the following sections can be useful in order to point out some of the basic issues underlying the behaviour of the studied actors[87].

3.1. b) The Scheme

First of all I will present a small introduction to the subjects of software development and Open Source. They will be used to create the setting in which the following discussion will take place and to point out some interesting questions about the differences between the Linux and Microsoft development paradigms. Then I will consider the different cultures and objectives of the analysed groups. After this, an analysis of some issues (decentralisation, communication and co-ordination, modularization, and testing) relevant to the management of complexity, will be presented. I am going to compare Linux and Microsoft practices in what refers to these questions, and assess which of the models can be expected to be more dynamically successful in the development of OSs. As we will see, all these issues are deeply related to that of motivation and objectives of the communities in charge of the development process. Once working hypothesis are generated, I will consider some facts about learning and imitation dynamics in the software industry as evidence about potential answers to the question of which development models and practices are more suitable for the production of OSs.

3.2: THE SETTING: DIFFERENT APPROACHES TO SOFTWARE DEVELOPMENT

3.2. a) Software development

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.

[87] I consider myself as quite a follower of Donald McCloskey´s methodological prescriptions, and therefore accept "good" science as comprised of facts, logic, metaphors and stories. In my opinion, quantitative data is not the only kind of empirical fact that can be used by the social scientist. See McCloskey (1990) for some methodological considerations.

-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 incoherences in their interaction.

What I have described so far constitutes what we could call the "traditional" or "waterfall" linear model for software development[88]. Although this methodology has been replaced by more modern practices that I will present in the following section, 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 the software project from its beginning. Given the complexity of software development, this certainty 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. Software development becomes, as Brooks states, "*the nightmare of its manager*".

The spiral model

The solution for many of these difficulties has been found in the "spiral model", followed, by Microsoft[89]. 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 we could consider as an iterative version of the waterfall model taking smaller steps. The boundaries between the different phases previously considered dissolves to a certain extent. 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 deal with as they are found and the decision about trade-offs between different functions becomes simpler. Each x 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 co-ordination and management of those groups in charge of different parts that constitute the system. In order to function correctly, this methodology requires a very well synchronised evolution of the components that integrate 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 assure a "perfectly reliable" product, given the wide scope of real situations and conditions in which it will be used (the external complexity mentioned in part 2). 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

[88] Se Brooks (op. cit. note 3) Pp. 265-266 and Cusumano and Selby (op. cit note 35) p. 192 for more detailed descriptions of this model and its associated problems.

[89] For an extensive description of Microsoft development processes see Cusumano and Selby (Ibid) Ch. 4 and 5. Also see Brooks (op. cit. note 3) pp. 271-272 for a consideration of this approach.

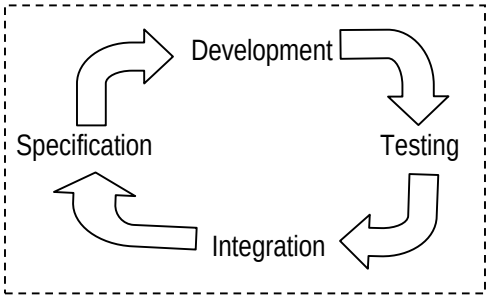
case of OSs, the most complex and difficult to develop software products. (see table 9 for schemes of the waterfall and the spiral development model).

TABLE 9- A sceheme of the waterfall and spiral models.

The waterfall model

Specification → Implementation → Integration and testing
(look for problems and solve them at this phase)

The spiral model



In n cycles until Beta testing in real conditions and release.
Look for problems from the beginning of the project and solve them as they are found.

3.2. b) 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 arrangement that are very different from those described before[90]. 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 because of strategic reasons that will be examined later.

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 own developers determine specifications and design of new features and functions. Any kind of change, improvement, correction or feedback about errors is submitted back to the "co-ordinating centre", who 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 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[91]. The numeration of the different releases

[90] For descriptions of the Linux development model see Moody (op. cit. note 10), DiBona, Ockman and Stone (eds)(1999) (book available on-line at http://www.oreilly.com/catalog/opensources/book/toc.html accessed 26/7/2001) and Raymond (Op. cit. note 57), specifically the essay "The Cathedral and the Bazaar", also available at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/ Accessed 26/7/2001

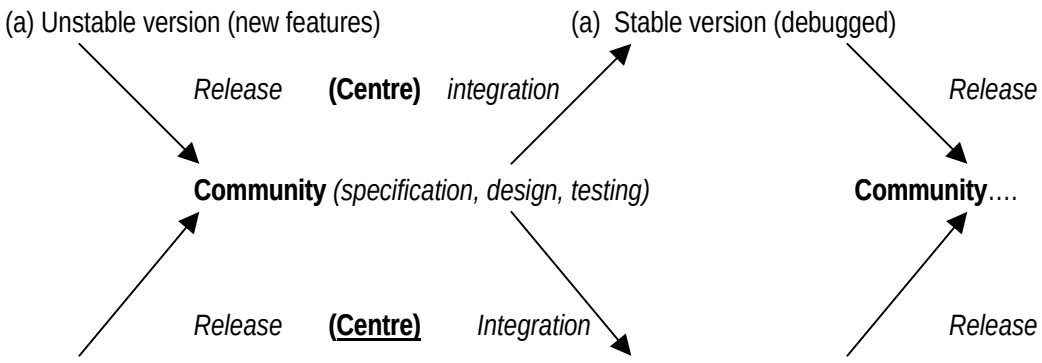
[91] Given the availability of the source code, anyone can try to fix the bugs found.

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[92].

This model is based on a very special kind of licensing arrangement, the General Public License (GPL), which covers all Linux releases. The GPL is a "copyleft" tool devised by Richard Stallman for his GNU (GNU´s not Unix) system, and states that "*users can copy and modify the program, and sell the original or modified versions, but they cannot modify or restrict the rights granted by the copyleft to any user*"[93]. 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. From a mainstream economics perspective, the problem with Linux would be that of most public property goods: an inefficiently small amount of the product would be created because of the problems in appropriating an income from the investment made in the product[94]. However, as I will discuss later, this does not seem to be the problem with Linux. Actually, the GPL is one of the reasons for volunteers helping in its development. This has to do with the motivations of the Linux Community, in which I will focus in the next section[95].

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 important uncertainties about the future evolution of the product.

TABLE 10: A scheme of the Linux development model



[92] Some users will be interested in a reliable and robust product for business use, for example. On the other hand, other users may want the latest release, with new features and functions, which maybe will be less reliable. About this releasing and numeration strategy see Moody (op. cit. note 10) pp 111-112 and Himanen (2001) pp. 66-67.

[93] Moody (op. cit. note 10) P. 26. In order to see the complete version of the GPL, see Free Software Foundation (wd) http://www.fsf.org/copyleft/gpl.html (accessed 26/7/2001).

[94] The question is why could an actor have reasons to improve the product. It seems as if there were very high incentives to "free ride" in the improvement of Linux.: wait for others to contribute and exploit their effort. If all actors worked on a profit-maximisation basis, nobody would rationally help in the development of Linux. The important point here is that of motivations for cooperating.

[95] Some important aspects of the GPL will be mentioned in the next part, when speaking about legal and institutional issues.

3.3- DIFFERENT PARADIGMS. CULTURES AND MOTIVATIONS

The main reasons that justify considering Linux and Microsoft products as being inside two different technological paradigms are the differences in the exemplars that they take as basis for their development, the methodologies followed during that development and the motivations and aspirations behind the actions of their constituents, in which I will focus now[96].

3.3 a) Culture and Motivations

I have already noted that each technological paradigm is comprised of people who do things. These people have motivations for their actions, and these motivations are determined by what we could define as "their culture"[97]. Now I will examine briefly some of the issues and differences between Microsoft's and Linux communities. Please bear in mind the fact that motivations and objectives constrain the kind of strategies, both technical and commercial, that actors can adopt.

Microsoft's Culture

We could define Microsoft's culture as one of technical excellence focused on the development of high quality products that are to be sold in order to increase market share and maximise profits[98]. Bill Gates, founder and leader of Microsoft, has been defined as a "*an organism which has been bred for the accumulation of great power and maximum profit*"[99]. In what refers 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"*[100]*.* It seems clear that for Microsoft, the development of the product is not an end, but just a mean for accomplishing an end, which is to gain money[101]. In this sense, the community inside Microsoft behaves according to Neo-classical maximisation postulates.

Linux Culture

The culture of those behind Linux is harder to define than that of Microsoft, and it goes against what, inside Neo-classical Economics, is defined as a rational behaviour. After all, the Linux community works for free: their product is developed by volunteers and distributed gratuitously. A very important question is what lies behind this seemingly altruistic behaviour. Some hypotheses and explanations have been proposed by different authors:

-According to Peka Himanen, the community behind Linux obtains satisfaction from the own act of working in the development of the product. In explaining these people's behaviour, Himanen uses what could be defined as a "craftsmanship model". There seems, according to literature on the subject, what could be called as "the joy of hacking" experienced by certain individuals[102].

---

[96] Some differences between the development models have been noted in the previous discussion, and will be analysed later. In what refers to the exemplars, "*basic artefacts to be developed and improved*" (Dosi (op. cit. note 19) p. 224), I will just say that Linux is based on the Unix OS created in the 1960s (See Salus (op. cit. note 1). In the case of Microsoft it is more difficult to determine which is the basic exemplar, and I would suggest MS DOS, its first product, as it was the genealogical antecedent of Windows, its current product. After all, Windows was built on top of MS DOS.

[97] Please note that culture is being used here as a very loose concept.

[98] Am considering profit maximisation in a dynamic, long-term perspective.

[99] John Seabruck quoted in Cusumano and Selby (op. cit. note 35).

[100] Ibid. pp 98-99.

[101] As Chris Peters, one of Microsoft´s top managers, stated, speaking about Microsoft: "*Your job is not to write code, your job is not to test, your job is not to write specs. Your job is to ship products…You´re trying not to write code. If we could make all this money by not writing code, we´d do it*" (ibid p. 45).

[102] Himanen (2001). See also Levy (1994). Hacking could, according to him, be defined as "a *project undertaken or a product built not solely to fulfil some constructive goal, but with some wild pleasure taken in mere involvement*" (p. 23).

-Eric Raymond, on the other hand, proposes a "gift economy" hypothesis. In his opinion, hackers contribute for free to the development of their product because in that way the increase their "status" amongst their peers, obtaining what he defines as "ego satisfaction" and increasing, therefore, their utility[103].

-I have also tried to contribute to the debate by elaborating what could be defined as the "information maximisation" hypothesis, according to which individuals inside Open Source communities have a "preference for information". As the amount of information produced by their community increases, so does their utility. They have incentives for sharing information, given the externalities that they start by doing so (they increase the potential for information-production amongst other members of the community, and therefore, the total amount created)[104]. In this sense, to collaborate in the development of Linux is to help in the creation of great amounts of technical information (both in quantity and quality)[105].

As the reader will have realised, these hypotheses are to a certain extent complementary (for example we could consider "status increase" in Raymond's terms as the flip side of the "information maximisation explanation: those who create the greatest amounts of information obtain status). In any case it is not the objective of my dissertation to determine the motivations of the Linux community (although I consider research on this issue as very relevant indeed).

In any case what seems clear is that the community of developers inside the Linux paradigm have the maximisation of the quality of their product as an intermediate objective, even if it may not constitute their main aspiration (which may be to enjoy their creative work, to increase their status or to maximise the amount of information produced).

We can think about the previous issue in the following terms: it seems clear that the Linux Community constitutes a group of individuals with shared interests: they like to develop software because of any of the previously considered reasons (that they do it for free and share the product of their work seems a strong demonstration of this). Somehow, by collaborating on Linux they engage in an activity that they enjoy. Linux (the OS) is, in a sense, the "focus" of this activity. That is, without Linux, they would not be able to perform it (their community would disappear or have to look for some other project). If these people want to keep doing what they do, Linux should survive[106]. As we have seen before, and given the existing network dynamics in the sector of OSs, the only way for the product to survive is to grow. And the only way to grow is by being competitive in the marketplace, by creating a high-quality product[107].

---

[103] For a more detailed presentation of this thesis, see "Homesteading the Noosphere" Raymond´s essay on the subject (op. cit. note 57), also available online at http://www.tuxedo.org/~esr/writings/homesteading/ . For a more political analysis with the same conclusion about gift economies, see Barbrooks paper, The High-tech Gift Economy, in which he establishes linkages between the Hacker and Open Source culture and Anarcho-Communists movements of the 60s. Available at http://www.socio.demon.co.uk/magazine/5/5barbrook.html#d3 accessed 26/7/2001.

[104] For a more detailed description of this hypothesis see Mateos García (op. cit. note 27). Kelly (op. cit. note 18) seems to adhere to the same kind of explanation for Open Source behaviours.

[105] Similar considerations could be used to explain information sharing by scientists. After all, many of Linux contributors are either students or academics. In the case of Linux we can consider information "embodied" in artefacts and technology.

[106] The Linux community shows a strong "confrontational" attitude towards Microsoft, that is considered an "enemy" in a very explicit sense. Sometimes this corporation seems to be seen as a threat to the "way of life" of those inside the Linux community.

[107] The second and the third Linux motivational hypotheses provide easier explanations of the drive for quality and growth: in the "status" case, the greater the community, the greater the potential status to be achieved by collaborating. In the second one (the information-maximisation one), the greater the community, the greater the potential amount of information produced.

Different ends and different means

The difference between Microsoft and Linux will by now seem clear. Not only are their development models different, but also the motivations. And we should take into account that these two questions are not independent. As I already mentioned in part 2, different objectives mean different constrains. I will later try to show how some of Microsoft's design and development decisions are constrained by the main objective of the firm, which is to maximise profits[108]. 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, reducing its potential profits. As Lessig, says, and we will see now in more detail, specific technological "architectures" embody specific "principles" and "objectives"[109].

3.4- ISSUES IN THE MANAGEMENT OF COMPLEXITY: AN ASSESSMENT

Having mentioned the interrelationship between motivations and potential strategies that can be adopted, I will focus now on specific issues having to do with how do the methodologies followed by each of the alternative paradigms being analysed 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.

3.4 a) 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[110]. 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 has also been signalled by Brooks as a way of improving software development[111]. 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 should be dealt with when altering the product.

Linux is a very modular OS, based on design principles inherited from the first Unix OS (one of the main fundamentals underlying Unix design philosophy is to *"build things that are for a single purpose but do that purpose well"*)[112]. Modularity is essential for Linux decentralised development model, as we will see later[113]. On the other hand, although Microsoft has introduced in its development process modularization design techniques, such as Object Oriented Programming[114], in many occasions has

---

[108] This discussion is based on the theoretical framework for the analysis of design proposed by Simon (1998), Ch. 5. According to him, when designing, a person chooses from all the "possible worlds" that meet environmental constraints (such as technical constraints in our case), the one that maximises his utility function. It is easy to see the differences between Linux and Microsoft developers in this picture.

[109] See Lessig (2000) (web document available at http://cyber.law.harvard.edu/works/lessig/pcforum.pdf accessed 26/7/2001). In the political sense discussed previously, also see Winner (1980), on the issue of artefacts embodying political values.

[110] This can be better understood by consideringa metaphor devised by Herbert Simon for the representation of this problem: imagine a watch-maker building a clock that needs a hundred pieces: it is easier to do it by creating ten small subassemblies composed of ten units each and then connecting those subassemblies between themselves than by trying to connect the hundred pieces together. See Simon (1996).

[111] Brooks (op. cit. note 3) p. 188.

[112] Salus (op. cit. note 2) p. 53

[113] See Torvalds (1999).

[114] For a non-technical definition of this programming technique see Kelly (1994) pp. 197-198.

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 (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[115].

3.4 b) Decentralisation

As I have said, 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 decentralisation is possible or not in 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[116]. Some reasons for this are the following:

-Through decentralisation is easier to determine the output of 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 firms[117].

-Through decentralisation "informational responsibilities" are more efficiently distributed across the firm. Following a argument similar to that of Hayek, and taking into account Simon´s theories on bounded rationality, we could say that the manager is to a certain extent relieved of the burden of co-ordinating the productive process to its finest degree (for what a enormous amount of information is required)[118]. Workers have more room for decision in the performance of tasks about which they are more 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 know better 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 (to a certain extent, we could say that "everyone can contribute to the innovative process")[119].

---

[115] A relevant example of this kind of integration is the one that was used to link Internet´s browser, Explorer, to the Windows Operating System: by interlinking both products´ source code, it was made much more difficult for consumers to remove the Explorer in order to use Navigator, the browser of Microsoft´s competitor, Netscape.

[116] Compare the fordist production line as a paradigm for the production of many commodities with the flexible production methods that have been lately introduced in many industries. It seems clear that the latter allows for a bigger degree of independency in the worker. For some examples see Valdaliso and López García (2000) pp. 464-474 and Gates (1999) Ch. 16.

[117] See Williamson (1975) for an analysis of this kind of problems, defined as "inseparabilities".

[118] See Hayek´s classical article on the advantages of markets over hierarchies on the management of information (Hayek (1945) and Simon (1957). I will have more to say about this issue when I speak about Communication and coordination in the next part.

[119] See Schumacher (1973) for a beautifully written discussion of this issue.

Brooks has already pointed out the advantages of decentralisation of software development in what he calls "the power of giving up power"[120]. Microsoft is one of the organisations that he signals as having developed a mentality of making "*large teams work like small teams*"[121]. 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 easier decentralised work (there is small need for co-ordination among great number of developers localised all around the world)[122]. The heterogeneity of these developers also provides Linux with what we could call a greater pool for the exploration of the space design, which increases the innovative potential of this system: Linux open nature makes it easier for many different people to examine its code and think about ways of improving it. Given the tacit, personnel 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[123]. Decentralisation makes it possible for literally hundreds of developers to concentrate in the same piece of code and explore different ways of improving and debugging it without an organisational hierarchy having to worry about "allocation of human resources". From those explored ways, 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 consensus amongst different projects inside an organisation. The independence between 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.[124]

Because of this same reason, and taking into account that Linux developers tend to be Linux users as well, we can 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, an important problem mentioned in part 2[125]

The main downfall associated to the kind of extreme decentralisation 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 that should be performed[126]. Although in some senses this

---

[120] Brooks, (op. cit. note 3) Pp. 277-279.

[121] Ibid (pp. 278-279). See also Cusumano and Selby (op. cit. note 35) p. 13. Bill Gates speaks about "*developing processes that empower people"* (Gates (Op. cit. note 109) Ch 16).

[122] This decentralisation becomes possible because of the expansion of the Internet. Kelly has already signalled (op. cit. note 104 Ch. 11) the influence of the web in organisational change, and I have mentioned it myself in part 1. Actually, modularisation appears as a requisite for the kind of decentralisation adopted by the Linux Community (see Torvalds, op. cit note 108).

[123] This is what Raymond (op. cit. note 57) calls a "collective exploration of the design space". Cusumano and Selby point out (op. cit. note 35) Pp. 420-421 that one of the potential problems of Microsoft is that the "*emphasis on "bang for the buck" (*commercial focus*) can discourage truly creative people*". The inflexibilities and rigidities sometimes created in the vision of those inside a certain paradigm (for example by an excessive commercial orientation of development) are weakened in the case of Linux by the non-exclusive nature of its community of developers.

[124] See Browne (wd) available at http://www.firstmonday.dk/issues/issue3_3/browne/ accessed 28/7/2001.

[125] Of course this does not solve the problem of designing for users needs, as "Linux developers" could be considered to form a social category with skills and aspirations different from those that can be found amongst the "public". Some people inside the Linux Community still considered this system as being designed *"by hackers for hackers"*, and this may affect its diffusion. On the other hand we should take into account that Linux is by now mainly being deployed as a Server OS, whose users are mainly technically skilled.

[126] Browne (op. cit note 114) refers to this problem and advocates for the creation of central organisations that sponsor particularly important development projects. He also mentions the problems that decentralisation causes in the provision of support for users. I shall refer to this point in the next part.

constitutes an advantage (self-assignation may mean an easy way of matching skills with tasks), on the other hand tricky and boring (but necessary) activities may be neglected[127].

3.4 c) 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 one 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" on which the community operates following the model sketched in table 10. The feedback that the centre obtains are the actual components that will be used to integrate the next release.

In the case of more formal management, there are decisions at the top which are transmitted to the bottom and then feedback from the bottom to the top, informing about the result 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 ask its subordinates to do it. Then he obtains information about the results. In contrast, in the first case, the "manager" just tells the community what the situation is and the community gives him the potential answers from which he 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 (there is a closer and more diverse contact with the technical milieu) and a more accurate perception (and conscience of the importance) of potential future alterations in the competitive, technical and social setting present[128].

Another important issue is that 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), 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 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 as the right ones.

---

[127] For a brief analysis on the virtues of self-assigning see Moody (op. cit. note 10) P. 62. Raymond (op. cit. note 103) states that in order to solve this kind of problems, the Linux Community assigns more status to those members who perform these ungrateful tasks. According to him "*continued devotion to hard, boring work (like debugging, or writing documentation) is more praiseworthy than cherrypicking the fun and easy hacks*".

[128] Think about the problems faced by Microsoft when having to adapt to the revolution of the Internet: Bill gates and most of the managers of the firm, focused on the development of the now almost completely forgotten "Information super-highway", neglected the advice from those workers and managers that thought that the Internet would be the next important change in the environment in which the firm was set. Once the managers perceived the relevance of the Internet, Microsoft showed a remarkable capacity for reorientation of its strategies (although we may ask ourselves if its come-back was not, to a big extent, made easier by its already accumulated power). See Wallace (op. cit note 73).

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.[129].

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, that I will illustrate 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 it 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 weaken considerably the strength and capacity of improvement of the product, with the network dynamics already mentioned creating a vicious circle of less developers, less quality, and therefore less developers…[130]. Forking can be caused by disagreements in technical questions, and its avoidance relies, to a great extent, in 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 apparently successful at this task[131].

3.4 d) Testing

I have already mentioned that testing inside a firm is unable to discover all the problems and bugs hidden in a program's code because of the impossibility of considering all 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 possible changes in 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*"[132]. The potential problems associated to this approach, that is, uncertainty about the degree of reliability of a product at a 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.

---

[129] See Lessig (1999a) (wd) available at http://cyber.law.harvard.edu/works/lessig/www9.pdf accessed 30/7/2001.
[130] Moody (op. cit. note 10) shows some examples in which a forking of Linux almost took place. See Ch 18 for a discussion of the issue and the reasons that make forking an unlikely case in Linux. Raymond (op. cit. note 103) also shows some ways in which the community discourages forking.
[131] This, of course, raises the issue of Linux dependence on its creator. On the other hand, similar questions could be asked about Bill Gates and Microsoft (see, for example, Cusumano and Selby (op. cit. note 35 P. 419).
[132] Raymond (op. cit. note 90)

TABLE 11: A summary of the advantages and disadvantages of the competing paradigms in the management of complexity.

Issue	Linux strategy	Linux advantages and disadvantages	Microsoft strategy	Microsoft advantages and disadvantages
Modularity	Modularisation	(+)It becomes easier to change and improve components of the product.	Modularisation and integration	(+) Commercial and strategic advantages of integration . (+) static improvement in performance (-) problems in the management of complexity and change.
Decentralization	Decentralised/ Parallel Development.	(+) Parallel exploration of the design space. (+) Incentives for optimisation of each of the components. (+) Adaptation of the product to different user needs (-) Allocation of resources to different tasks.	Limited decentralisation	(+) Allocation of resources to the performance of tasks that are perceived as necessary. (-) Limited vision of potential solutions and improvements. (-) Limitations in the potential adaptation of the product to user needs.
Communication	Non-hierarchical bottom-up approach	(+) Less need of a formal management of information flows (+) Adaptability of the product to needs perceived by the community and changes in the environment. (+) Integrators commitment to quality. (-) Possibilities of forking.	Hierarchical top-down approach	(+) Control over the development of the product by the management (-) Bounded rationality of managers (-) Organisational inertias. (-) management of information Flows upstream and downstream.
Testing	Testing in real conditions.	(+) More comprehensive testing. (+) Assured robustness of stable versions of the product. (+) Direct communication of users with developers.	Internal testing and Beta testing.	(-) Less comprehensive testing. (-) Less reliability in new products. (-) Need for management of the information flows between users and the organisation.

(+) signals advantages of the strategy and (-) disadvantages.

3.4 e) A summary: change versus control

In table 11 I have summarised some of the questions discussed during this section. What seems to appear is a dichotomy between two different ways of organisation, one that favours change and other that strives for control. In my opinion that first model, followed by Linux, could be placed close to what Kelly has defined as the "swarm model", a network of autonomous units that work following simple rules. According to him, the advantages of this model are adaptability, evolution, resistance, lack of boundaries and novelty (innovation). Its disadvantages would be that it is non-optimal, non-controllable, non-predictable, non-understandable and non-immediate[133]. Kelly considers very different phenomena inside this category, which he defines as the most efficient ones in dealing with complexity in a fast-changing environment where static efficiency in moment t means inefficiency in the moment t+1, when everything has changed.

[133] For definitions of these terms see Kelly (op. cit. note 114).

Of course Linux development model and organisational arrangements cannot be considered to fall completely into the swarm model, since some hierarchical mechanisms that are used to co-ordinate productive activities can be found in its structure. Still it seems closer to that model than Microsoft. After all, Microsoft needs a control over the product in order to obtain profits: integration and bundling, management of teams and scheduling appear as strategically important activities, used for commercial reasons. Linux, given its not-for-profit nature, does not have to face these constraints. The community can employ development models that seem more suitable for the creation of high quality software.

Closed source

From the need of management and control with the objective of maximising profits stems Microsoft's use of a closed source strategy, which constitutes its essential difference with Linux. While it is true that Application Interfaces (the OSs ports to which applications connect) are open and available to external developers, the internal functioning of the product is hidden. To keep the source code of its product secret and proprietary determines, in last instance, Microsoft's control over its product and the possibility of extracting profits from it. The source code can be used in an strategic way to keep different programs integrated, to enjoy a competitive advantage in the development of complementary products such as applications, and to avoid anyone using that code to build upon it potentially competing products[134]. This need for control is determined by the objectives of Microsoft, that as I showed before are commercial and constrain the potential development strategies available for the company. Since Linux is not subject to this constrain, and control of the product is not a relevant issue for its community, open source strategies, that make possible the adoption of the previously described methodologies with their associated advantages, can be used. The issue of culture, aspirations and incentives could be considered as essential factors in the determination of the technological possibilities available for the competing paradigms.

Constraints and objectives as focusing devices

Focusing devices constrain the unfolding of a technological trajectory and influence the channels towards which innovation and incremental improvement of a product are directed. In the case of software development I mentioned that the constrains to which the community of developers is subject, whether caused by its own motivations and objectives or by external causes (such as complexity and fast rates of change in the software industry), influence the kind of progress that may take place along the technological trajectory. In this part I have been focusing on how the different motivations of the actors being analysed constrain the technical and organisational strategies they may adopt, conditioning, therefore, the final results of their development efforts, that is, the efficiency of their technological paradigm in producing a "dynamically high-quality" product. While both Linux and Microsoft are subject to the kind of "external constrains" mentioned in section (c) of the previous part, in the case of the motivational constraint, we find important differences between both groups, that affect the potential strategies that they may adopt. We find how the profit-maximisation objective creates relevant constraints in the kind of technical and organisational arrangements that can be adopted by Microsoft, limiting, therefore, the efficiency of its development model in improving and problem-solving activities, associated to what we can consider as "the unfolding of a technological trajectory".

[134]While, as I said, APIs are theoretically open, some commentators have accused Microsoft of placing in its OS "hidden APIs" only available for other Microsoft developers, who enjoy an advantage in the creation of applications that make use of the resources of the OS. See http://www.zdnet.com/anchordesk/stories/story/0,10738,2595479,00.html accessed 22/08/2001 for an ZDNet article on this issue.

3.5- LEARNING AND ADAPTATION

The basic theme underlying the previous discussion has to do with the fact that some of the development strategies available for Linux cannot be adopted by Microsoft, even when being more suitable for the creation of high-quality products, because of the constraints created by its profit-maximisation objectives. We can observe, however, how Microsoft, in all the issues analysed previously, has adopted strategies that, in a limited way, are parallel to those followed by Linux. This gradual adaptation has taken place through learning during the development process and the interaction with users. Following Rosenberg's terminology we could consider Microsoft as introducing innovations in its development process in order to deal with the kind of bottlenecks and problems that reduced its productivity and efficiency in the development of high quality, competitive products. The interesting aspect of this kind of evolutionary learning is that it has, in many cases, driven Microsoft practices and methodologies close to what I have described as the Linux model, and could constitute evidence hinting towards an inherent superior efficiency of the Linux development model[135].

After all, and as I mentioned in section (d) of part 2, even through learning and change in the development strategies adopted by actors do take place, there is a basic constraint in the scale to which they may be carried forward, determined by the motivations and objectives of the actors. Microsoft faces an essential trade-off between the development methodologies that maximise quality and those that maximise profits. These kind of trade-offs limits the possibilities for adoption of a Linux-esque development model, where, as I will mention later, profit possibilities are much smaller.

In my opinion, Microsoft development methodologies and processes constitute the benchmark for commercial software development. But commercial software development (that is, the one carried forward by profit-motivated firms) does not constitute the whole world of software development, as the existence of Linux shows. Being devoid of the kind of constraints faced by Microsoft, the Linux Community can adopt a series of strategies more suitable, for the development of high quality, dynamically improving and innovative software products. In this section I will briefly consider some of the parallelisms between Linux methodologies and those adopted, in a limited way, by Microsoft[136].

3.5 a) Development, modularization and decentralisation

I have already mentioned how Microsoft has tried to modularise its products and decentralise the operation of its teams. These type of strategies are necessary for the use of the synch-and-stabilise methodology, that could be considered as a smaller scale version of the Linux development model, and has solved many of the problems that the company faced in the past: different developers work on components independently and then integrate them in order to be able to solve problems as they appear. Products are built incrementally and the "internal releases" are tested immediately. Product specifications are defined in open, flexible ways, leaving room for change during the implementation process. This way, more creativity and independence in developers´ work is allowed. In order to alleviate the problem of meeting deadlines, "buffering" time is left at the end of the development process with the aim of allowing project teams to deal with unexpected problems. In any case we can find interesting parallelisms between Microsoft and Linux models: Microsoft has tried to follow the principles of modularization and decentralisation, but to a lesser extent than Linux.

---

[135] In what refers to Linux, I would say that many of its practices are derived from previous hacker and scientific methodologies, very efficient in the creation and diffusion of knowledge See Himanen (op. cit. note 102) Ch. 4. We could also analyse Linux growth and expansion through the concept of emergence, that is, the "generation of complex organised systems by a small number of rules". See Kelly (ibid) and Holland (1998)

[136] The following discussion is going to be based on the description of learning and evolution of development practices in Microsoft, by Cusumano and Selby (op. cit. note 35).

Their adoption by Microsoft is, after all, limited by the need to keep the source code inside the company, the integration requirements for bundling and the need to write *a priori* specifications in order to define "products". The same could be said about the need for hierarchies and control. Microsoft has tried to avoid bureaucratisation and to foment independence in the teams by establishing product units with their own budgets and clear communication channels with the top management. But total decentralisation and the application of developer's creativity to the improvement of the product are limited by the need to keep schedules and maintain coherence between the strategies of different units. Limitations of manpower (workers are Microsoft's main asset, as its managers state, but also do constitute a cost) and time (given the importance of "time to market" in the software industry) condition the extent to which modularization and decentralisation may take place, limiting the extent to which their associated advantages may be enjoyed. See table 12 for a synthesis of this discussion.

TABLE 12: Limitations for Microsoft in the adoption of development strategies: modularisation and decentralisation.

LINUX MODEL	MICROSOFT LIMITATIONS	MICROSOFT MODEL
-Modularisation: building models from independently developed components. -Decentralisation: independence between productive units. Parallel exploration of the design space. -No a priori specifications: they are defined by the developers.	-Integration as a commercial strategy -Need to keep source code closed. -Budget constraints: personnel limited. -scheduling constraints: time resources limited. -strategic coherence: need to coordinate the development of different programs to build products.	-Synch and stabilise processes (small teams coordinating their activities and integrating components into internal releases). -independence of teams. -More flexibility through open product specification and buffering time. -…but all of this subject to managerial control.

3.5 b) Usability and testing

Microsoft employs diverse techniques in order to incorporate user feedback into the development process. Amongst these are the use of market research, usability labs, activity based planning or product usage studies. These methods were devised as a reaction to usability problems found in Microsoft's first products. In any case, these techniques cannot be as comprehensive as the widespread distribution of the product for real use through fast releases and the incorporation of feedback into the development process for the next release. Instead of defining a user and then developing a product around his "image", in the case of Linux the user defines himself and adapts the product to his needs. This question becomes especially important in cases where customisation of a system may be necessary in order to adapt it to user requirements. In order to deal with this kind of problems Microsoft builds products with many functions that users can configure in different ways. The use of this strategy is, again, determined to a big extent by profit maximisation objectives: Microsoft tries to develop products targeted at big markets, and does not have many incentives to create specialised products (or allow users to customise them)[137]. Linux users, on the other hand, and given the availability of the source code, can simply alter the product in order to adapt it to their needs (or hire someone to do it)[138].

[137] This creates the problem of "creeping featuritis" mentioned by Brooks (op. cit. note 3) p. 258, introduces additional complexity into the development process and may increase difficulty in the use of the product.
[138] The fact that Linux has been ported to so many different hardware architectures is one of the results of this freedom of customisation.

In the case of testing, we observe clear parallelisms between the testing of internal releases and Beta Testing (used by Microsoft) and Linux testing methodologies. Microsoft tries to improve the reliability of its product by having it used in real conditions, both by its developers (one of the principles of Microsoft is that workers should use the company's products) and external individuals. Again, as in previous cases, Microsoft's use of this technique is limited by the need to keep a schedule and control over the product[139]. Putting in practice Beta Testing in the large scales observed in Linux would be very difficult for Microsoft, and in any case the need to keep control over the source code makes it impossible the kind of thorough testing, debugging and problem-solving that may be carried out by Linux users.

3.5 c) Shared Source

Microsoft is known for of its use of what has been called as "embrace and extend" tactics. In a broad sense this means adopting competitors strategies and innovations that have proved successful in the past and using them in the competition with their creators[140]. These imitation dynamics are common in all industries, and form part of the externalities associated to innovation. In the case of Linux strategies, Microsoft's proposal is what has been defined as "Shared Source"[141]. Shared Source grants licensed users access to certain parts of Microsoft Shared Source. The Shared Source License, at the same time, tries to protect Microsoft's Intellectual Property Rights, "*needed to support a strong software business*". Given the kind of objectives and intentions of Open Source communities, and their aversion to Microsoft, it seems unlikely that this strategy will be able to create the kind of committed response that has been observed in the case of Linux[142].

In any case, the Shared Source initiative, as an imitation dynamic, seems a clear acknowledgement by Microsoft of some of the advantages inherent to open source strategies. As Microsoft states, its adoption is limited by the need to keep a "business", with its profit-maximisation connotations.

3.6- CONCLUSION

In my opinion, the adoption of the Shared Source strategy by Microsoft constitutes a summary of the argument that I have been developing during this part of the dissertation: Open Source development models present strong advantages when compared with Microsoft ones, given the nature and problems associated to software development, but Microsoft cannot employ them without damaging its profitability. The differences in objectives cause differences in strategies, and therefore, in results.

However, as has been shown in the theoretical model, this quality, which includes the attribute of compatibility, may be influenced by the structure and dynamics of the software industry, limiting the competitiveness of the product. I will focus on these issues in the next part of my dissertation.

---

[139] During the browser wars, Microsoft imitated Netscape´s rapid beta release rate of its browser (see Cusumano and Yoffie (op. cit. note 50) pp. 282-287). It does not seem probable that this technique would be adopted for OSs release (usually distributed by Microsoft through Original Equipment Manufacturers or retail stores) , and in any case the unavailability of the source code would diminish the potential benefits of this strategy.

[140] Some examples would be the development of the Graphic User Interface as a response to Apple or the embracing of internet protocols and methodologies used by Netscape.

[141] Netscape and Apple have already imitated Open Source methodologies through the creation of Mozilla, an open source browser, and Darwin, an Open Source OS. See http://www.mozilla.org/ and http://www.apple.com/darwin (accessed 29/7/2001) for more information.

[142] See http://www.microsoft.com/business/licensing/sharedsource.asp accessed 29/7/2001 for an overview of the subject. Some members of the Linux community see the Shared Source initiative as a way of fragmenting the Open Source world. See, for example, , http://www.shared-source.com/ and http://www.perens.com/Articles/StandTogether.html (accessed 29/7/2001) for assessment of the Shared Source iniciative by Open Source groups.

4- STRUCTURES, INSTITUTIONS AND INTERVENTION

In this part I will consider certain issues having to do with competitive strategies and dynamics in software industries that can influence and even determine the result of competition between the different products. In the previous part I tried to show some theoretical and behavioural evidence that points towards the superior efficiency of the Open Source technological paradigm in the development of high quality software products. However, and no matter how many literature on complexity or problems associated to software projects I use, I acknowledge that it is the market (that is, consumers), and not an assessment of strengths and weaknesses, the one who should demonstrate the relative efficiencies and quality of competing products. My concern is that in the case of software, and specifically Operating Systems, the market seems to be dominated by one of the contenders, who can use certain strategies mentioned in section (e) of the previous part to interfere with competition and favour its products irrespectively of quality issues. In this part I will focus on those strategies that may be used by incumbent actors in order to gain market advantage, and may provide a rationale for Public intervention in the OS market.

First I will present briefly the Linux business model, as an introduction, and refer to some questions that could diminish the competitiveness of the Linux OS, taking into account the special characteristics of the software industry. Afterwards I will present some historical evidence about the strategies that can be used by incumbent actors to gain advantage in competition in software markets, referring then to the specific case of the competition between Linux and Microsoft. I will use for the discussion the theoretical framework and concepts laid out in 2.2.(d). Finally I will present some final considerations about the policy implications of the analysis carried forward during my dissertation.

4.1- THE OPEN SOURCE BUSINESS MODEL

I have mentioned before that Linux is available for free in the Internet, and can be downloaded, together with its source code, by anyone interested in it. This is an essential part of the Linux development model. But given the special characteristics of the software industry, this “distribution” methodology has some problems that have been partially solved by the creation of what we could call a “Linux business model”, that will be briefly described in this section. But before doing that I will refer to those issues that have reduced Linux competitiveness. They are, mainly, support and cultural questions.

4.1.a) Support

I have mentioned before the fact that software users are mainly concerned about the “dynamic quality of a product”, that is, its evolutionary improvement and enhancement possibilities. Inside this “dynamic quality” category we could consider the issue of support, that could be defined, in a broad way, as “problem solving”. Given the essential unreliability of most released software products, many users (especially business users) want to be able to receive help in case they have problems with the product. This help is provided by support and customer help services made available, in many cases, by the firm who develops the product. In the case of Linux, support would take place through the Internet, that is, a Linux user with a problem would use the web to communicate with the Linux community and obtain a solution. But this kind of arrangement seems quite problematic, in part because of trust issues that have much to do with the kind of cultural questions that have been considering before.

4.1. b) Culture and trust

Support is has much to do with credibility, which as I said in section (d) of part 2, is deeply related to trust. I also mentioned that trust has much to do with culture: individuals tend to trust more those with the same kind of views they have.. In the case of the Linux support model, there seems to be a problem of users' mistrust on Linux Community's commitment to the provision of support for their OS. The argument could be posed in the following terms: "the Linux community is not legally obliged to support the product they release. This creates uncertainty amongst potential users about the support they will receive when using Linux". This mistrust seems to have a clear rational basis (as, for example, users of mission-critical software will want guaranteed help in case they have a problem), but in my opinion is also linked to cultural issues: If we consider the business community (which as I said before is the main demander of software support) as having, by definition, a "profit-maximisation" culture, mistrust towards a development model which is not based in those "profit-maximisation" objectives could be expected[143]. Of course I am posing the question in simplistic terms, but there is some evidence about these kind of dynamics: it has been observed that in some business, engineers (that we could consider as having a more "technical" background, closer to Linux developers' own) have started using Linux in secret, hiding this from their managers, who do not favour "free" software, and prefer to use commercial one [144].

In the theoretical model, we could consider these kind of effects as reducing the scale of the network dynamics associated to the use of the Linux OS and restricting its adoption. There is a problem with the "perceived dynamic quality" of the Linux OS, which has been at least partially dealt with with the creation of the Linux business model.

4.1. c) Public Relationships, support and business

The problem of Linux support has created a business opportunity. Some firms, such as Red Hat, Caldera or Debian have appeared as intermediaries between users and the Linux Community of developers. They commercialise distributions of Linux, that is, tested and integrated releases of the freely available product, warranted and supported by them[145]. This diminishes the uncertainty suffered by business users, because they find, in these companies, someone who can (using Bryan Sparks, former CEO of Caldera words) "*stand up and take the bullet*" (if there are problems with the program)[146]. Following the previous discussion, we could say that these firms act as a cushion that alleviates the conflict between the different cultures found at each of the sides of the spectrum depicted in table 13[147].

One of the advantages of this business model is that, contrary to Microsoft, and because of the use of the General Public License, nobody owns "Linux", and therefore technology lock-in, with the associated problems for consumers and potential entrants, becomes much more difficult. There seems to be something close to potential perfect competition between different Linux Distributors[148].

---

[143] The business user may consider, from his "profit-maximisation" point of view, that a "not-for-profit" company cannot be trusted as a profesional solution for his software problems.

[144] See Moody (op. cit. note 10) Pp. 100-101 for some examples.

[145] These business could also be considered as a intermediary between technically skilled and knowledgeable developers and less "technically interested" users. One of the main service provided by these companies, is, for example, an easier installation of the product.

[146] Ibid. p. 101

[147] The kind of model I am refering to is defined by Raymond as "Give Away the Recipe, Open a Restaurant". See Raymond (op. cit. note 57)

[148] ibid for a discussion of these issues

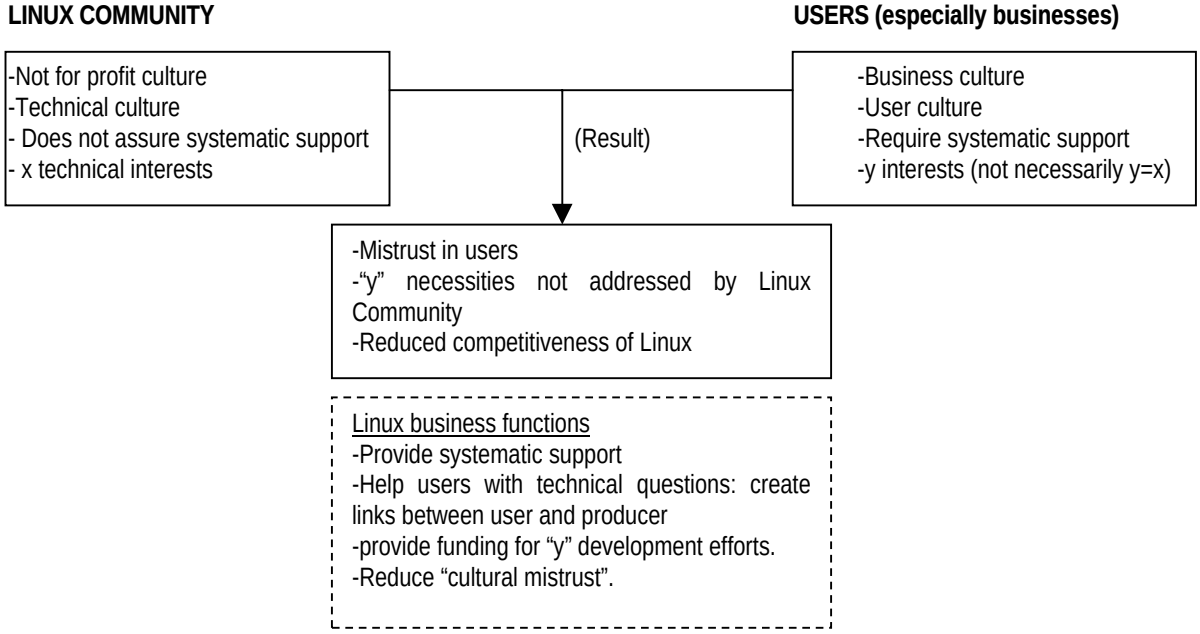
On the other hand, the main problem associated to this model is the need to maintain compatibility between the Linux releases made by different firms[149]. Some of them may have incentives to betray what we could call Linux standards in order to capture market share. After all, competition inside open standards reduces the profit opportunities for firms, as I mentioned in 2.2.d)[150].

Patronage and co-ordination

The Linux business model is based on the fact that the costs of acquiring the OS in which these firms base their activities is almost null (Linux is available for free in the Internet). These firms depend on the effort of the volunteers of the Linux Community. One way of paying for the Linux Community's contribution to their business model is providing financial support to certain development efforts.

Here again Linux firms may be seen as intermediaries between the Linux Community and its users. Firms may perceive user needs that are not being addressed by the Linux community (maybe because the differences in objectives that exist between both groups) and communicate those needs to the developers, together with financial help that will provide incentives for the performance of effort in those necessary projects. This, to a certain extent, may contribute to solve the problem of "assignation of efforts to tasks" mentioned in the previous part[151].

TABLE 13: Some functions of the Linux business model



[149] This was one of the main problems associated to Linux' predecessor, Unix: it was fragmented in many different versions that suffered compatibility problems reducing the size of its network of users and the intra-market and inter-market network externalities associated. See, for example, Cusumano and Yoffie (Op. cit. note 50) Pp. 125-129 for the problems found by Netscape when trying to provide product support to the different versions of Unix.

[150] "*In my opinion, the GPL is optimized to build a strong software community at the expense of a strong commercial software business model. That's why Linus Torvalds said last week on ZDNet that "Linux is never really going to be a rich sell.*"" (Craig Mundie's-Microsoft Senior Vice-President- speech on CNET available at http://www.microsoft.com/business/licensing/ssmundie.asp accessed 21/8/2001). A solution for this problem is the proposed creation of a "Linux standard setting institution" called The Linux Standard Base. See Moody (op. cit. note 10) Pp. 310-311 and http://www.linuxbase.org/ accessed 24?8/2001.

[151] See Raymond (op. cit. note 57) for an analysis of the behaviour of Linux firms as "patrons" for the Linux Community.

4.2- INCUMBENT STRATEGIES

In section (e) of the theoretical model I mentioned three possible loci where different strategies may be adopted by the competing actors. The technical choices, having to do with the development models followed by each of the technological paradigms, have been discussed in the previous part. Now I will focus in the business strategies that may be used, mainly by incumbents, in order to gain advantage in their competition with potential entrants into the market: I will analyse some of Microsoft's observed behaviours, placing them in the theoretical model previously devised, and explain how can they affect the competitiveness of alternative products, interfering with competition in the OS market. I will also refer to the third channel mentioned in that section, having to do with the importance of the institutional environment, and mainly the Intellectual Property regime.

4.2.a) legal and technical obstacles to compatibility

I have said that one of the strategies that can be used by potential entrants in a market dominated by one standard is to achieve compatibility with it. However, there are technical and legal strategies that may be used by the incumbent player in order to interfere with that potential compatibility and reduce the inter-operability possibilities of new competitors.

Technical strategies

In order to achieve compatibility with an existing network, potential entrants will need certain technical information about the communication protocols that it follows. Open source protocols are freely available and anyone can use them in order to create devices able to communicate with the network that has adopted them. On the other hand, in the case of closed source and proprietary protocols, the potential entrant will need either to ask the incumbent for information or reverse-engineer its system. The first possibility will be implausible in many cases, since the incumbent will have small incentives to allow entrance of other players into its network.. Andrew Tridgell, creator of Samba, a basic server that allows Unix (including Linux) and Windows machines to share files and communicate with machines operating with Microsoft Windows NT OS points out that many people inside Microsoft have tried to hinder his efforts in following the evolution of Microsoft's communication protocols[152].

Legal strategies

Intellectual property Rights over components of a system may be used to avoid competitors connecting to an existing network. I said before that the second possible strategy for the acquisition of information about the communication protocols used by an incumbent network is reverse engineering. This kind of action may be impeded by existing laws, such as those expressed in the Digital Millennium Copyright Act, which prohibits the "unauthorised access to copyrighted work" through reverse engineering[153]. The legal framework may appear as impeding the entrance of new competitors in the proprietary network.

Credibility

Apart from denying access to information necessary to guarantee compatibility between different products and using Intellectual property Laws to impede the use of reverse engineering strategies, an

[152] Moody (op. cit. note 10) p. 274

[153] There are any provisions that allow reverse engineering in order to make inter-operability between different systems possible, but in which cases is this strategy allowed seems uncertain. See Linux Today reports on how the Digital millenium Copyright Act makes it ilegal for Linux OS to support DVD systems. http://linuxtoday.com/stories/15655.html accessed 23/8/2001. See also Lessig (1999b) wd p. 764-765.

incumbent may also spread uncertainty about the compatibility of a new entrant with the dominating system.

Many authors have argued that this was one of the strategies followed by Microsoft with DR-DOS, a now disappeared OS that competed with MS-DOS . According to Wendy Goldman Rohm and James Wallace, Microsoft "FUDded" DR-DOS by introducing in Windows 3.1 artificial message errors that sprang when it was used over the DR-DOS, and by making statements about "*compatibility of DR-DOS with MS-DOS not being guaranteed*" [154]. By creating fear in users about their possibilities in connecting with the Microsoft network, Microsoft gained advantage in the market.

Dynamic compatibility

In addition to the previously mentioned questions, we should remember that the achievement of compatibility takes place in a dynamic sense, as communication standards and protocols evolve. The need to keep track of the evolution of the proprietary standard over time as its owner alters it adds difficulty to the task of guaranteeing inter-operability with the incumbent technology[155]. These alterations may be used in a strategic way, with the incumbent trying to turn its standards into a "moving target" for the potential entrants, and making it difficult for them to assure compatibility with the existing network.

Extending and polluting Standards

In addition to managing the potential compatibility of new entrants' technology with its own, dominant one, an incumbent may use its market power in order to extend its standard-setting power to other environments, raising the kind of barriers for competition that have been mentioned previously. In the famous Halloween Papers, assumedly written by Microsoft engineers (the company has been neither accepted nor denied this affirmation), it is stated that one of the strategies that could be potentially used against Open Source models is that of "de-commoditizing" standards[156]. I have already mentioned this strategy of "pollution" as one that affects open standards, turning them into closed ones through the creation of proprietary extensions, and making it possible for firms to capture market share. In the case of the Java programming Language, Sun has accused Microsoft of polluting its communication standards by adding proprietary, closed extensions to the software and creating incompatibilities with Sun's version[157]. It has also been argued that Microsoft is trying to pollute the open communication protocols of the Internet. This kind of strategies, that according to Moody, have been used with Kerberos, an open security protocol, would enable the company to control the technology of certain sections of the Internet, determining the connectivity potential of other systems such as Linux, and "*denying Open Source Projects access to the markets*" (in the author of the Halloween Papers own words)[158].

4.2. b) Industrial structures and inter-market network effects

I said in part 2 that the creation of industrial structures, albeit essential for the integration of the systems of which OSs are just a component, may also become a barrier that impedes the entrance of competition to a market. New entrants may, through different strategies that will be mentioned next,

---

[154] See Goldman Rohm (op. cit note 73) Ch. 7 and Wallace (op. cit. note 73) Pp. 52-55.
[155] Mansell and Steinmueller (op. cit note 12) Pp. 296-297
[156] For a version of the Halloween Papers commented by Eric Raymond see http://www.opensource.org/halloween/ accessed 22/8/2001.
[157] See a collection of documents about the Sun vs Microsoft lawsuit at http://java.sun.com/lawsuit/filings.html accessed 24/8/2001
[158] Moody (op. cit. note 10) Pp. 311-312. See also http://www.usenix.org/publications/login/1997-11/embraces.html accessed 24/8/2001 for a description of the "embrace process". Given its power, Microsoft can use this kind of "embrace and extend" tactics in order to appropriate competitors' technology which is freely available in the Internet, such as in the case of Linux, and turning it into a proprietary standard that could be sold by them.

have the access to essential complementary resources such as hardware or applications denied. This loss of compatibility with vital components of the system will weaken the competitiveness of those products, favouring those who already are in dominant positions. In this section I will mention some of the strategies available for incumbents in this field.

Compatibility and complementors

Shapiro and Varian define complementors as "complementary components" of a system with which a product should be compatible in order to be competitive at all. Co-ordination with complementor producers in networked markets is essential for the integration of a product in broader systems[159]. Different strategies may be used in order to deny potential entrants to a market access to these complementors' resources: for example, Microsoft non-disclosure arrangements in the past prohibited those companies that were developing products for Windows to work in similar products for any competing OS during the period of one year. Although these arrangements were prohibited by the 1995 Consent Decree, there is evidence about Microsoft still using these tactics in order to avoid applications being developed for competing products, with all the associated inter-market network effects[160]. Another example of this behaviour would be the delays in the development of applications for Apple in which Microsoft incurred. By slowing the access and quality of applications for this platform, Microsoft was diminishing the strength of Apple in its competition with Intel (Microsoft´s main partner). Certain design decisions (mainly, to design applications for PCs and then adapt them with minimum customisation for Apple Computers), impaired Apple´s competitiveness *vis à vis* the Wintel platform[161].

The intimate collaboration between industrial partners may also provide advantages to incumbent players, for example by closing for potential entrants access to information about new technologies. For example, until recently, information about Intel's new chip architectures were made available for Linux Developers months later than for Microsoft's ones. This greatly reduced the competitiveness of Linux products, always lagging behind their better-informed rival[162].

Closing access to distribution channels

In addition to reducing the potential compatibility of competing products with the resources developed by the partners of the incumbent player, existing industrial structures may also exclude potential entrants from distribution channels, diminishing the availability of their product and its potential adoption. For example, exclusive arrangements with Original Equipment Manufacturers (who sell PCs with an already installed OS) may make it impossible for other OS developers to install their software in the hardware manufactured by these OEMs, denying them access to the market. The use of this kind of practices was one of the reasons for Microsoft's legal problems during the early 1990s[163]. Although the already mentioned 1995 Consent Decree prohibited some of these arrangements, Goldman Rohm argues that the company has kept using many of them on a more subtle way. Given the company's power and importance, intimidation and strangling tactics may be used to force OEMs to incorporate Microsoft's software into their products instead of that developed by competitors[164].

---

[159] Shapiro and Varian (op. cit. note 7) P. 10

[160] Many observers have considered Microsoft buying of Corel's stock in October 2000 as a strategy aimed at elliminating this company's support to Linux. See Moody (op. cit. note 10) P. 297 and byte.com analysis of the investment at http://www.byte.com/documents/s=475/BYT20001010S0005/index2.htm accessed 20/8/2001.

[161] See Cusumano and Selby (op. cit. note 35 p. 150-151)

[162] Moody (op. cit. note 10) Pp. 286-287

[163] See Gilbert (1995) for a review of this casen and an analysis of how Microsoft's Licensing practices affected competition.

[164] For the case of Microsoft's arrangements with Vobis, the German OEM, see Goldman Rohm (op. cit. note 73) Ch. 5

Industrial arrangements and market power

Industrial partnerships are necessary for the systemic integration of the different components of Information and Communication Technologies, but when their members act in discriminatory ways, they may become a barrier for competition. Incumbent, powerful players may use their inter-sector linkages and relationships in a strategic way, turning industrial arrangements into collusive pacts. Access to resources, information and distribution are some examples of how these strategies may be used in order to diminish the competitive potential of new entrants into the OS market.

What could be said as a conclusion is that although many practices employed by Microsoft in the past have been ruled out as illegal by the courts, the company still has enough market, financial, and therefore, intimidating power to exert a notable influence over producers of complementors in ICT markets. This influence can be used in order to close the doors for potential entrants into the OS market[165].

4.3- THE INTELLECTUAL PROPERTY REGIME

In section (b) of Part 2 I said that the legal framework constitutes a very important part of the environment in which technological paradigms are set and act. By determining whether the objectives of the constituencies behind these paradigms may be achieved through the performance of their productive activities, it may influence in very important ways their incentives and behaviours. In this section I am going to point out an important issue related to the Intellectual Property Regime (IPR), that affect the development models of the competing paradigms, that is, the question of information appropriability[166].

I have already said that the Linux model is based on the free release of information, mainly embodied in the source code that constitutes the Linux OS. Elsewhere I have spoken about how the possibility of appropriating this kind of publicly released information may make it impossible for models such as Linux to survive: actors with profit-maximisation preferences may simply take the information freely released by others and appropriate it. The existing IPR, based on the assumption that individuals need economic incentives in order to produce information, and devised with the aim of providing those incentives through patenting and copyright systems, may eliminate the possibility of alternative behaviours and motivations[167]. The tool that makes the use of those free-riding, appropriating strategies difficult and the survival of Linux possible is the existence of the General Public License (GPL), mentioned previously. This fact makes recent attacks on this license relevant, in the sense that since it makes the Linux Development Model sustainable, alterations or limitations on its scope and depth could affect the model's strength in developing high quality software products.

According to Craig Mundie, Microsoft's Vice-president, the GPL turns Intellectual property Rights "upside down" and makes a commercial software model inviable[168]. He views the GPL as a threat to the IPR, basis for the whole software industry[169]. He is mainly concerned with certain elements of the

---

[165] See Brown (op. cit note 2) Pp. 90-91 for a parallel argument.

[166] I have already spoken about how by making the use of reverse engineering ilegal in some cases, the legal framework may close access to the technical information necessary for potential entrants to achieve compatibility with incumbent, dominant networks. The extension of patenting rights to ideas such as "programming techniques" may also create obstacles to Linux development (Moody (op. cit. note 10 P. 311). See Mansell and Steinmueller (op. cit. note 12) for an analysis of the downfalls of strengthening IPR in what refers to these issues.

[167] See Mateos Garcia (op. cit note 27) for the full analysis.

[168] See Mundie's article published at ZD.Net News, available at http://www.zdnet.com/zdnn/stories/comment/0,5859,2761605,00.html accessed 28/8/2001.

[169] It is interesting that these concerns have not appeared until Linux has reached high levels of market share. It also seems clear that the modification of the GPL would pose a threat for the survival of Open Source Communities. We can observe in this case the conflicting interests and groups.

GPL that make it possible to infect the intellectual property of firms, turning it into "open source"[170]. Although this matter is controversial, the strategy followed by Microsoft seems to be related to the kind of indirect strategies aimed at altering the legal framework, the third potential loci for action mentioned in the theoretical model[171].

This kind of strategies seem to follow the statement made in the Halloween Paper, referring to the fact that "Linux should be targeted not as a business, but as a process (by Microsoft)". It seems that strategies that deprive this model of its base, the GPL, may be useful in order to limit its growth and diffusion.

4.4- IMPLICATIONS

In the previous sections I have briefly reviewed some of the potential strategies that may be used by incumbent players (in this case Microsoft) to diminish the compatibility, availability and competitiveness of new entrants in the OS markets. It seems clear that given Microsoft's power, the company may exert a considerable influence over future developments in ICT markets and industries. Instead of predicting the future, the company has reached a size with which it may *determine* it. For example, in the past Microsoft was able to redirect the tide that pointed towards the establishment of Internet Browsers as the main interface through which computers would be used, and eventually turned these into applications subordinated to OSs[172]. In cases where different technological paths are available, and given Microsoft's profit motivations, it may not be assumed that its decisions will coincide with those that can be considered as the more desirable from broader, public perspectives. After all, and as I have tried to show during this part of the dissertation, the adoption and success of technologies is not directly determined by their efficiency, but also by institutional, structural, legal and social factors such as those mentioned before. When we consider this in conjunction with the analysis of the relative efficiencies of Microsoft's and Linux software development models that took place in the previous part, we reach some important implications that will be presented in this section.

4.4. a)- The rationale behind policy

The story that I have told so far contradicts the kind of "competitive process" arguments that describe Microsoft as a monopolist that has been more efficient than its competitors[173]. I have shown evidence that points towards the superiority of Linux Development Model in the creation of high quality products, and also some strategies and reasons because of which this superiority will not necessarily result in effective competition against Microsoft. The mechanisms and structures mentioned in this part may become barriers too high even for products that best Microsoft's ones, and seem to define a clear justification for public intervention in the OS market[174].

4.4. b) Four Guidelines for policy

In my opinion, the playground and the rules of the game followed in ICT industries, were created and have evolved following very clear assumptions about which behaviours are efficient and conduce to innovation and growth. These assumptions are framed as cultural biases that favour profit-

---

[170] "*If software released under the GPL is combined with proprietary code, the resultant combination has to be released under the GPL*" (Moody op. cit. note 10 Pp. 26-27). One of the apparent objectives of this clause is to avoid of the use of "embrace and extend" tactics that would turn Open Source into proprietary by combining it with proprietary extensions.

[171] This strategy could also be understood as a way of introducing uncertainty about Linux and reducing the incentives for its adoption and use. See the GPL analysis FAQ at http://www.microsoft.com/business/licensing/sharedsource.asp accessed 28/8/2001.

[172] See Goldman Rohm (Op. cit. note 73) Ch. 16.

[173] This kind of story can be read, for example, in Stross (1996).

[174] The assumption underlying this reasoning is that, basically, competition favours innovation.

maximisation behaviours and mistrust alternatives. As I have argued in different points of this dissertation, there is to a certain extent a cultural clash between what the conventional wisdom defines as "correct" ways of performing productive activities, and the actual efficiency of the different possible models. This mistrust generates ostracism: Industry consortia close their doors to groups with these different motivations, the creators of Intellectual Property Laws do not listen to their voice and the consumers feel fear, uncertainty and doubts about their products. This way, conventional expectations become self-fulfilled and the way that one is expected to behave becomes the only possible way of behaving. And if, as is the case of OS markets, in addition to these problems, that alternative has to face a rival as strong as Microsoft, it seems that the possibilities of surviving and growing are really small. Linux has been able to do so, and propose its model as an alternative, both in commercial and cultural terms. The implications of this are far reaching, but if we are to see them realised, maybe some action that guarantees a fair competition in the OS market will be necessary. As a conclusion for this part I would like to suggest four guidelines for policy related to the issues mentioned in this part:

-1) Measures that favour compatibility and point towards an easier achievement of inter-operability of new technologies with the dominant ones should be implemented. Competition only becomes possible when networks, to a certain extent, fuse, and substitution of new products for entrenched ones becomes possible. Measures should be taken so that the kind of technical information related to inter-operability and compatibility issues be made more readily available, and those strategies that seek incompatibility and the pollution of standards as a way of capturing markets are discouraged.

-2) In order to achieve co-ordination between different organisations in the determination and evolution of standards, it seems desirable to achieve a greater openness in industry standard-setting consortia. The kind of discriminatory arrangements present in closed industry structures interfere with competition, and it seems necessary to assure a broader participation in these forums. This would constitute a clear acknowledgement of the "public good" nature of standards. Improvements in the performance of public standard setting organisations would be useful[175].

-3) The fact that, even when facing the kind of barriers previously mentioned, Linux has been able to survive and thrive as an alternative to Microsoft's products, points out the importance of assessing carefully potential alterations in the legal environment that may damage the capabilities of its Open Source Development model. It seems necessary to attenuate the existing tendency towards strengthened Intellectual Property Rights signalled by Mansell and Steinmueller[176]. As this dissertation has shown, behaviours not based on profit motivations are not only possible, but in some cases, superior to the business ones. The success of Open Source should bring a rethinking on the limitations and objectives of IPR.

-4) Public expenditure and adoption of Open source products can be used in order to increase their critical mass and enlarge their user networks, providing incentives both for incumbents and complementors to grant compatibility of their own products with them. Given their relative cheapness, Open Source products could be used for the implementation of Universal Access policies[177].

[175] See Mansell And Steinmueller (op. cit. note 12) Ch. 6 for some issues related with public standard-setting organisation and industry consortia as different ways of determining standardisation.
[176] Ibid. Ch. 7
[177] For an assessment of some potential advantages of Universal Accesspolicies see ibid. Pp. 240-256.

5- CONCLUSION

In this last part I will present a brief list of relevant issues that have not been dealt with directly in this dissertation, propose some questions for further research and finish with some personal considerations.

5.1- OTHER RELEVANT ISSUES AND PERSPECTIVES

The main analytical concept that underlies this dissertation is that of efficiency, defined in a dynamic sense, and this of course limits the scope of the work. In this section I will mention briefly some alternative perspectives from which the analysis of competition between Microsoft and Linux should, in my opinion, be conducted.

5.1. c) Macro perspectives

The micro definition of efficiency that I have adopted in this work, related to the suitability of different development models for the creation of a product in an environment of complexity, leaves out of the discussion many macroeconomic issues that should, in my opinion, be considered when assessing the relative advantages and disadvantages associated to each of the technological paradigms that constitute the target of this research. Questions such as the social costs associated to technology lock-in to a closed standard or the necessity of technological diversity as a value in itself, that are related to broader definitions of social efficiency, should, in my opinion, be taken into account when discussing the need for transparent competition between Linux and Microsoft.

5.1. a) Technology and democracy

Even if an analysis employing broader definitions of efficiency had been provided for, there would still be the need to consider additional values that should be appraised when analysing Linux and Microsoft as competing technological paradigms.

In the third part of the dissertation I provided the reader with evidence that seems to suggest the superior efficiency of the Linux development model when compared with Microsoft's one, and I said that this superiority stems from the different motivations underlying each of the technological paradigms. Microsoft's maximisation objective (developing software as a way of obtaining a profit) limits the range of strategies that can be adopted by the firm in the construction of its product and, therefore, the eventual success of its development effort. But as Lessig states, "*However efficient open code may be, arguments about open source must also consider the questions that these values rise. For, in my view, it makes as much sense to promote open source on efficiency grounds alone as it does to promote democracy on grounds of economic wealth alone. It may well be that democracies are more wealthy than other forms of government, just as it may well be that open source software is more robust than others. But this is a thin conception of value that would see wealth or efficiency as the only, or most important, value at stake*[178]". Lessig argues that the Open Source movement embeds certain values related to the ways of developing and using technology that are more democratic, transparent and inclusive than those inside the closed source model. His analysis seems close to the propositions of the Constructive Technology Assessment school, that consider necessary the democratisation of technological processes as a way of increasing not only the efficiency, but also the suitability and legitimacy of new technologies.

[178] Lessig (Op. cit. note 153) P. 769

Although my analysis has taken place in efficiency terms, I think that the consideration of additional values is essential when assessing new technologies, and that a comparison of Microsoft and Linux in these terms should take place[179].

5.1. b) A Marxian Analysis

The different motivations underlying Linux and Microsoft productive activities may bring us back to the Marxian concept of alienation. According to Marx, alienation appears when work stops being an end (a task that is natural in the human being, and therefore good) and becomes a mean (a necessary evil that should be borne in order to obtain money, which becomes the end). Alienation can be, in simple terms, understood as a state of confusion and unhappiness faced by workers during the Capitalist phase of history. In certain senses, we could analyse Linux and Microsoft motivational aspects in parallel to these two poles, that of work as an end and work as a mean. Many of Marx's concepts and dichotomies seem applicable to the comparison between Linux and Microsoft as different production modes, and interesting results could be obtained, in my view, from the use of this perspective.

5.2- QUESTIONS FOR FURTHER RESEARCH

In the previous section I have alluded to new perspectives that could be used to analyse competition between Linux and Microsoft and appraise their advantages and disadvantages. In a sense, they could be understood as broad propositions for further research. Now I want to present a more narrowly defined set of topics, whose future research I consider relevant for this subject. These are just a few from those that may come to mind. After all, this dissertation is concerned with the exploration of a subject that forms part of a nascent Programme of Research, and in most of the topics that are touched, a deeper analysis seem necessary. In any case, the following issues, are in my opinion especially interesting.

5.2. a)- Exemplars and development models

In this dissertation I have mainly focused on the organisational aspects of the development models adopted by each of the technological paradigms being considered, neglecting to a certain extent important technical aspects. Research on how do the technical characteristics of the exemplars, that constitute the basis for the development effort, influence the strategies that can be adopted, and how is the design of these "seminal" artefacts related to the objectives of the constituents of the paradigm is, in my opinion, important.

5.2. b)- Motivations of the Linux community

In part 3 I mentioned some hypotheses that have been posed by different authors when trying to explain reasons underlying the behaviour of Linux and other Open Source Communities. More extensive research on this issue is, in my opinion, necessary, in order to define more clearly the kind of factors that enter into the motivations of this social group.

5.2.c)- Economics of Technical Change and Linux

A very interesting topic future for research would be an analysis of the Linux community using the concepts developed inside the Economics of Technical Change. This kind of work would show us where the limitations and assumptions underlying this theoretical framework lie, maybe also pointing out ways of improving it and making it more complete. In my opinion we should ask of it the capability

[179] We could consider it roughly as an analysis from an "equality" perspective, the other value with which Economic Science was clasically assumed to be concerned.

of explaining, or at least making sense of the behaviours of productive units, not just "profit-maximising" (or satisfying) ones.

For example, one important aspect of the Linux model that has not been explicitly addressed during this dissertation is the importance of the Internet as a communication and co-ordination tool used by the Linux Community. This brings forward questions having to do with transmission of information and knowledge, as well as its use, through modern telecommunications systems. Further research on the advantages and disadvantages of Linux's seemingly codified knowledge transmission model appears as relevant.

Associated to this previous topic would also be that of research on the processes through which learning and accumulation of competencies take place inside Linux, and the determination of whether these concepts are useful for the analysis of the Linux and other Open Source Communities .

5.2.d) Empirical aspects

In addition to all these subjects, there appears a need for data on the development and adoption statistics of the Linux OS. Enquiries in this subject appears as necessary if analyses on this topic are to adopt a new quantitative dimension, and meaningful numerical comparisons between Linux and other development models are to take place.

5.3- LAST CONSIDERATIONS

"Whereas the Elizabethans were poised between medieval corporate experience and modern individualism, we reverse their pattern by confronting an electric technology which would seem to render individualism obsolete and the corporate interdependence mandatory"

Marshall McLuhan[180]

In my opinion the most important aspect of this dissertation is the emphasis on the cultural aspects of technological development and how do they influence and constrain the design and construction of technological artefacts. Culture, an issue that has not been thoroughly considered in economic analysis, becomes an essential question that explains the main differences between the considered technological paradigms. In that sense, we can understand this dissertation as being, in an abstract sense, concerned with why and how do people do what they do, with their engagement with technology as an economic and creative activity.

Technology is a basic influence in cultural evolution, and to a big extent the subject of this dissertation appears as a result of the diffusion of a new technology, the Internet, that has made it possible for cultural alternatives to the dominant production models to appear and thrive.

As we know, cultures clash and collide, as conflicts between different conceptions of the world, different aspirations and objectives appear. This dissertation seems to be examining one of those cases. Cultural competition takes place in the marketplace, but not only there. I have spoken about power, institutions, laws and policies; these factors are very important, and their evolution does not seem to follow a "natural path". Decisions, influenced by ideological perspectives and biases, are taken. One of the intentions of this dissertation has been to argue that since those contingent decisions shape technological and cultural evolution, maybe a consideration about the relative social desirability of the alternative values and paradigms, should take place.

Juan Mateos Garcia 29/9/2001

[180] McLuhan (1962) p. 1

BIBLIOGRAPHY

LITERATURE

Alchian, A. A. and Allen, W. R. (1964): *University Economics*. Belmont, Wadsworth Publishing Company.

Arthur, W. (1989): *Competing Technologies, increasing returns and lock-in by historical events.* The Economics Journal 99, 116-131.

Auster, P. (1997): *The Art of Hunger.* London: Faber and Faber.

Battram, A. (1998): *Navigating Complexity: The Essential Guide to complexity Theory in Business and Management.* London: The Industrial Society.

Baumol, W., Blackman, S. and Wolff, E. (1989): *Productivity and American Leadership*. Cambridge, Massachussetts: The MIT Press.

Baumol, W. (1992): *Perfect Markets and Easy Virtue: Business ethics and the invisible hand*. (Spanish Edition, (1993): *Mercados Perfectos y Virtud Natural. La Ética en los Negocios y la Mano Invisible*. Madrid: Celeste Ediciones)

Blaug, . (1980): *The Methodology of Economics.* Cambridge: The Press Syndicate of the University of Cambridge.

Boehm, B. and Ross, R.(1989): *Theory-W Software Project Management: Principles and Examples.* IEEE Transactions on Software Engineering, Vol. 15, No. 7, July.

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

Brooks, H. (1986): *The typology of surprises in technology, institutions and development* in W. Clark, C. Munn (eds.): *Sustainable Development of the Biosphere.* Cambridge, Cambridge University Press.

Brown, D. (1997): *Cybertrends: Chaos, Power and Accountability in the Information Age.* London: Viking/Penguin.

Brynjolfsson, E. (1993): *The Productivity Paradox of Information Technology*. Communications of the ACM, Vol 36, No. 12, December. pp 67-75.

Callon, M. (1992): *The Dynamics of Techno-economic networks* in Coombs, R., Saviotti, P. and Walsh, V. (eds.): *Technological Change and Company Strategies.* London: London Academic Press.

Cusumano, M. and Selby, R. (1995): *Microsoft Secrets: How the World´s Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. New York: The Free Press.

Cusumano, M. and Yoffie, D. (1998): *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft.* New York: The Free Press

David, P. (1975): *Technical Choice, Innovaion and Economic Growth: Essays on American and British Experience in the Nineteenth Century.* Cambridge: Cambridge University Press.

David, P. (1986): *Understanding the Economics of QWERTY: The Necessity of History* in William N. Parker, ed., *Economic History and the Modern Economist* . Oxford: Blackwell.

David, P. (1995): *Standardization policies for network technologies: the flux between freedom and order revisited* in Hawkins, R., Mansell, R. and Skea, J.: *Standards, Innovation and Competitiveness: the Politics and Economics of Standards in Natural and Technical Environment.* Aldershot: Edward Elgar.

DiBona, C., Ockman, S. and Stone, M. (eds.) (1999): *Open Source: Voices from the Open Source Revolution.* Beijing: O´Reilly and Associates.

Dosi, G. (1984): *Technical Change and Industrial Transformation.* London: McMillan Press.

Dosi, G. (1988): *The Nature of the Innovative process* in Dosi, G., Freeman, C., Nelson, R., Silverberg, G. and Soete, L. (eds.): *Technical Change and Economic Theory.* London: Pinter.

Eggertsson, T. (1990): *Economic Behaviour and Institutions*. Cambridge, Massachusetts: Cambridge University Press.

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

Freeman, C. (1992): *The Economics of Hope: Essays on Technical Change, Economic Growth and the Environment*. London: Pinter

Friedman, M. (1953): *Essays on positive Economics*. Chicago, University of Chicago Press.

Gates, W. (1999): *Business @ the Speed of Thought.* London: Penguin Books.

Gibbs, W.W. (1997): *Taking Computers to Task.* Scientific American, July, pp. 64-71.

Gilbert, R. (1999): *Networks, Standards and the Use of Market Dominance: Microsoft* in Kwoka, J. and White, L. (1999): *The Antitrust revolution: Economics, Competition and Policy.* Oxford: Oxford University Press.

Goldman Rohm, W. (1998): *the Microsoft File. The Secret Case Against Bill Gates.* London: Random House

Greenbaum, J. and Kying, M. (editors) (1991): *Design at Work: Cooperative Design of Computer Systems.* Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Hayek, F. (1945): *The Use of Knowledge in Society.* The American Economic Review, 35, 519-530.

Hill, C. (1997): *Establishing a standard: Competitive strategy and technological standards in winner-take-all industries:* Academy of Management Executive, Vol. 11 No. 2.

Himanen, P. (2001): *The Hacker Ethic and the Spirit of the Information Age.* London: Secker & Warburg.

Holland, J. (1998): *Emergence: from Chaos to Order.* Oxford: Oxford University Press.

Kelly, K (1994): *Out of Control: The New Biology of Machines, Social Systems and the Economic World*. Reading, Massachusetts: Addison.Wesley.

Kelly, K. (1998): *New Rules for the New Economy*. London: Fourth State.

Kuhn, T. (1996): *The Structure of Scientific Revolutions.* Chicago: The University of Chicago Press.

Landauer, T. (1995): *The Trouble with Computers: Usefulness, Usability, and Productivity.* Cambridge, Massachusetts: The MIT Press.

Lecht, C. (1977): *The Waves of Change: A Techno-Economic Analysis of the Data Processing Industry.* New York: McGraw-Hill Book Company.

Levy, S. (1994): *Hackers, Heroes of the Computer Revolution*. London: Penguin.

Mansell, R. and Steinmueller, E. (2000): *Mobilising the Information Society: Strategies for Growth and Opportunity.* Oxford, Oxford University Press.

Mateos García, J. (2001): *Can Information be Free?: Motives and Evolution of the "Information should be Free" Principle.* SPRU spring term paper.

McCloskey, D. (1990): *If You´re So Smart. The Narrative of Economic Expertise.* Chicago: University of Chicago Press,

McLuhan, M. (1962): *The Gutenberg Galaxy: The making of typographic man.* London: Routledge.

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

Paull, M. (1995): *The Evolution of the SEI´s Capability Maturity Model for Software*. Software Process- Improvement and Practice, Pilot Issue, 3-15.

Pavitt, K. (1911): *Key Characteristics of the Large Innovating Firm.* British Journal of Management. Vol. 2, 41-50.

Raymond, E. (2001): *The Cathedral and The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Beijing: O´Reilly and Associates.

Rip, A., Misa, T. And Schot, J. 1996): *Managing Technology in Society: The approach of Constructive technology Assessment.* London, Pinter.

Rosenberg, N. (1976): *Perspectives on Technology.* London: M.E. Sharpe.

Salus, P. (1994): *A Quarter Century of Unix*. Reading, Massachusetts: Addison-Wesley

Schumacher, E. (1973): *Small is Beautiful: Economics as if People Mattered.* New York: Haper and Row.

Shapiro, C. and Varian, H. (1999): *Information Rules: A strategic guide to the Network Economy.* Boston, Massachusetts: Harvard Business School Press.

Simon, H. (1957): *Administrative Behaviour. A Study of Decision-Making Processes in Administrative Organisations.* New York: MacMillan.

Simon, H. (1998): *The Sciences of the Artificial.* Cambridge, Massachusetts: The MIT Press.

Smith, A (1776): *An Inquiry into the Nature and Causes of the Wealth of the Nations.* Dent Edition (1910).

Sterling, B. (1993): *The Hacker Crackdown: Law and Disorder in the Electronic Frontier.* London: Viking.

Stross, R. (1996): *The Microsoft Way: the Real story of how the company outsmarts the competition.* London: Little and Brown.

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

Valdaliso, J. and López García, S. (2000): *Historia Económica de la Empresa.* Barcelona: Crítica.

Wallace, J. (1997): *Overdrive: Bill gates and the race to control cyberspace.* New York: Wiley.

Williamson, O. (1975): *Understanding the Employment Relation: The analysis of Idiosincratic Exchange.* The Bell Journal of Economics, 6, 250-278.

Winner, L. (1980): *Do Artefacts have Politics?*. Daedalus 109, 121-136.

Winograd, T., and Flores, F. (1986): *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, New Jersey: Ablex Publishing.

Web pages and documents (wd)

Barbrook, R: *the High-Tech Gift Economy*.
Web address: http://www.socio.demon.co.uk/magazine/5/5barbrook.html#d3

Browne, C: *Linux and Decentralised development.* Web address: http://www.firstmonday.dk/issues/issue3_3/browne/

David, P. (1986): *Understanding the economics of QWERTY: The Necessity of History.*
Web Address: http://www.stanford.edu/group/mmdd/SiliconValley/David/QWERTY.html

DiBona, C., Ockman, S. and Stone, M. (eds.) (1999): *Open Source: Voices from the Open Source Revolution.*
Web Address: http://www.oreilly.com/catalog/opensources/book/toc.html

Lessig, L. (1999a): *Cyberspace´s Architectural Constitution.*
Web Address: http://cyber.law.harvard.edu/works/lessig/www9.pdf
Lessig, L. (1999b): *The Limits in Open Code: Regulatory Standards and the Future of the Net.*
Web Address: http://cyber.law.harvard.edu/works/lessig/BerkPub.pdf

Lessig, L. (2000) : *The Code in the Law and the Law in the Code.*
Web Address: http://cyber.law.harvard.edu/works/lessig/pcforum.pdf

Free Software Foundation: *GNU General Public License.* Web Address: http://www.fsf.org/copyleft/gpl.html

Mundie, C. (2001): The Commercial Software Model and Sustainable Innovation.
Web Address: http://www.microsoft.com/business/licensing/ssmundie.asp

Pavitt, K. (2000): *Innovating Routines in the Business Firm: what matters, what´s changing and what´s staying the same?*. SPRU Electronic Working Papers.
Web Address: http://www.sussex.ac.uk/spru/publications/imprint/sewps/sewp45/sewp45.pdf

Raymond, E. *The Cathedral and the Bazaar:* Web Address: http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

Raymond, E. *The Magic Cauldron*. Web address: http://www.tuxedo.org/~esr/writings/magic-cauldron/

Raymond. E. *Homesteading the Noosphere*. Web address:
http://www.tuxedo.org/~esr/writings/homesteading/