Embedding knowledge in Enabling Environments: Internet, Web, GNU/Linux, Debian

Alberto Cammozzo*

* System manager, Department of Stastistics, University of Padova. Email: mmzz -at- stat.unipd.it
Paper last revision: 07/04/07Pages: 30

Abstract: *This analysis aims to show how each of four projects (Internet, the Web, GNU/Linux and the Debian project) establishes an environment embedding layered knowledge coming from individuals, firms and communities. All those actors cooperate and compete inside the environment and contribute to its building. Factors common to all those projects (as product and process modularity and openness), are especially effective and enabling for "peer production". Evolving in such a context gives free/open source software a competitive advantage.*

Keywords*: software value chain, knowledge value chain, software modularity, layering, software life-cycle, components, peer-production, enabling environment, strategy, layered environment, free software, open source, f/oss, Internet, Debian, Web, GNU/Linux, distributions.*

Introduction

This paper is divided in three parts: the *first* examines some specific aspects of Internet, the Web, GNU/Linux and the Debian project. The common aspects between the projects will then be highlighted, suggesting the emergence of a common pattern: an environment where other projects are likely to develop in an incremental and layered way and where feedbacks are reinforcing the underlying architecture.

The *second* part will deal with Internet as an infrastructure for software development, examining the relation between software life-cycle and the network through Porter's *value chain* framework. The result will suggest how the environment made by Internet and the Web not only *enables* free software, but gives it a competitive advantage. The importance of choosing the correct business model for commercial projects will also emerge.

The *third* part will recall how modularity and layering are the key aspects that make large software projects manageable. The process of building a GNU/Linux distribution will provide an example on how knowledge is being embedded along a specific value chain, providing added functionality and complexity simplification.

All along this paper the concept behind the terms *enabling environment* will become more precise, wrapping together technological innovation, social, economic and competitive activities with knowledge production. The conclusions will suggest that this is the kind of environment where more and more innovation activities will take place.

I – Some key aspects of Internet, The Web, GNU/Linux and the Debian project

Practical revolution is made of two things:
proof of concept and running code
Eben Moglen, – *Die Gedanken Sind Frei*

The trick for managers and programmers is to create
enough structure to keep projects under control but not so
much that "the process" stifles creativity and flexibility
Michael Cusumano – *The business of software*

The recent communication revolution is based both on the network of networks called Internet and on the Web, the hypertext and multimedia information delivery system which lays over it. The way the net behaves depends on how this architecture is built. *Free/open source software* (f/oss) has been at the center of this revolution since the beginning, not only exerting a technological influence but also shaping the building process (Benussi 2005).

One of these free software products, the GNU/Linux operating system, was born from a distributed, Internet based community of programmers. It is in fact based upon two partially overlapping and competing projects: the GNU (*GNU is Not Unix*) operating system and the Linux kernel. They eventually combined in one modular project, although still keeping their individuality.

The Debian project builds and maintains, on voluntary base, one of the most appreciated and successful *distributions* of the GNU/Linux operating system.

A number of common key elements can be found between Internet, the Web, GNU/Linux and the Debian project:

- they are *modular,* which makes cooperative work easier;
- they are *layered,* having being incrementally built on successive, progressive technological beds;
- on the boundaries between layers and modules, *open standards* an have been designed;
- they have been developed from a loose form of *cooperation* between different subjects, based more often on a set of common understandings and protocols than on central directions;
- coordination process and project governance are open, based upon a set of documents that define a *mission* and provide clear procedures for a shared decision making process focused on consensus among all stakeholders;

- these different projets can themselves be layered, concurring to create an *enabling environment* which can be a fertile ground on which other projects can grow;
- although these projects are no-profit, they carry relevant economic value, provide an opportunity for business, cooperate and compete on the market like firms, often following a strategy.

In order to prove those statements, for each project this section will analyse: (1) competitive factors, (2) layering and modularity, (3) governance and coordination, and (4) foundation principles and emerging strategies.

1 – Internet

Competitive factors

Internet imposed itself[1] on other established geographical networks: SNA (IBM), DECnet (Digital), Bitnet (open academic architecture), and also *peer-to-peer* networks, like Fidonet or UUCP. There are different reasons for this success. The openness of TCP/IP protocols on which Internet is based had a major role: those were already present on every UNIX workstation and server, then very popular in universities. Another key factor came from cooperation: with TCP/IP it was easy to aggregate and interconnect different networks, without costly central coordination from some proprietary firm. Although other complex architectures were technically superior, being simple and thanks to its flexibility Internet was able to deliver the same level of service and performance of proprietary networks and gradually took the place of its competitors.

Layering

Layered architecture is common to any advanced computer network. Each layer represents an abstraction that offers services to an upper layer without the need to know the implementing details of lower layers. Internet defines only two central layers (IP and TCP), taking advantage of existing technologies for the lower (physical) layers, while more complex network architectures (as the ISO standard) count up to 7 layers. This simple design and the independence from physical infrastructure allows its adoption without costly investments.

Modularity

Internet can be defined as a network composed by independent and autonomous networks that can therefore be seen as an independent module: successive modules can be added incrementally. Modularity can also be found in its architecture: based on the underlying two layers, each network service (as email, file transfer, printing, web browsing etc.) has its own independent protocol. On the whole, those services offer the upper layer of the Internet. Yochay Benkler sees modularity as a key aspect of Internet and many other

1 Very interesting insights on the history of Internet and the Web can be found in Ilkka Tuomi (2002).

network based projects (Benkler 2006):

> *The core characteristics underlying the success of these enterprises [Internet and peer-production] are their modularity and their capacity to integrate many fine grained contribution.*

The fine granularity is seen as a key factor for project manageability.

Cooperation, coordination and governance

Each network composing the Internet is independently administered, in respect of general interoperability rules and standards. Some important central coordination activities are assigned primarily to two entities: the Internet Engineering Task Force[2] (IETF), who evaluates and makes protocol proposals official, and the *no profit corporation* Internet Corporation For Assigned Names and Numbers[3] (ICANN), regulating domain names and assigns network numbers. Even if technically independent, practically ICANN depends on the United States government, and its status is currently under debate.

IETF, whose home organization is the Internet Society[4], promotes the standards on which Internet is based. Its most important role is regulate the way layers and modules are bound together through interfaces and protocols. Ad-hoc working groups address specific projects, coordinated by a *steering committee*. Inside each group, consensus drives the decision-making process (Bradner 1998):

> *The core rule for operation is that acceptance or agreement is achieved via working group "rough consensus". [...]*
> *IETF consensus does not require that all participants agree although this is, of course, preferred. In general, the dominant view of the working group shall prevail. (However, it must be noted that "dominance" is not to be determined on the basis of volume or persistence, but rather a more general sense of agreement.) Consensus can be determined by a show of hands, humming, or any other means on which the WG [Working Group] agrees (by rough consensus, of course). Note that 51% of the working group does not qualify as "rough consensus" and 99% is better than rough. It is up to the Chair to determine if rough consensus has been reached.*

There is a significant difference between Internet governance infrastructure and traditional coordination of organizations such as the ITU (International Telecommunications Union). A good synthesis can be found in an ITU report (Hayashi 2003):

> *In order to maintain the integrity of the global telecommunications network, standard setting, rule making and monitoring of the system is necessary on the global scale. Such a*

2 On IETF see: http://www.ietf.org/

3 On ICANN see: http://www.icann.org/general/

4 Om ISOC see: http://www.isoc.org/isoc/

task has been performed by ITU, ICANN and IETF. ITU is administered as part of the United Nations on the traditional international legal framework, i.e., its membership comprises individual sovereign nations, decisions are made on the unanimity basis, and the agreements are concluded as international treaty between ITU and the participating nations.

On the other hand, the internet technology requires new forms of global organization. For example, the Internet Corporation for Assigned Names and Numbers (ICANN) handles IP address space allocation, protocol parameter assignment, domain name system management, and root server system management functions. The Internet Engineering Task Force (IETF) is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the internet architecture and the smooth operation of the internet.

The outstanding characteristics of these new organizations are that they take individuals as members. No corporations or countries have representation in them. They are not inter-national organizations since they do not act on nations. They are trans-national organizations because ***they involve concerned individuals*** *across countries and regions. Also, they use a non-traditional approach in which participation and compliance is only voluntary.*

Principles and strategies

Even being very informal[5], IETF has formal rules. Similarly to firms, IETF and many other entities base their strategic action on *statements* and principles, sometimes explicitly formulated as a *mission* or *vision.* IETF goals and objectives are well defined in the document *A mission statement for the IETF* (Alvestrand 2004):

The ***goal*** *of the IETF is to make the Internet work better.*

The ***mission*** *of the IETF is to produce high quality, relevant technical and engineering documents that influence the way people design, use, and manage the Internet in such a way as to make the Internet work better. These documents include protocol standards, best current practices, and informational documents of various kinds.*

The same document draws the process and identifies resources to serve the purpose while remaining in the scope of the mission; this kind of behavior can be seen as a strategy (Alvestrand 2004):

The IETF will pursue this mission in adherence to the following ***cardinal principles****:*

Open process *- any interested person can participate in the work, know what is being*

5 As an example of "formalized informality", this is the *dress code* di IETF:
Dress Code: since attendees must wear their name tags, they must also wear shirts or blouses. Pants or skirts are also highly recommended. Seriously though, many newcomers are often embarrassed when they show up Monday morning in suits, to discover that everybody else is wearing t-shirts, jeans (shorts, if weather permits) and sandals. http://www.ietf.org/tao.html#2.3

decided, and make his or her voice heard on the issue. Part of this principle is our commitment to making our documents, our WG mailing lists, our attendance lists, and our meeting minutes publicly available on the Internet.

***Technical competence** - the issues on which the IETF produces its documents are issues where the IETF has the competence needed to speak to them, and that the IETF is willing to listen to technically competent input from any source. Technical competence also means that we expect IETF output to be designed to sound network engineering principles - this is also often referred to as "engineering quality".*

***Volunteer Core** - our participants and our leadership are people who come to the IETF because they want to do work that furthers the IETF's mission of "making the Internet work better".*

***Rough consensus and running code** - We make standards based on the combined engineering judgement of our participants and our real-world experience in implementing and deploying our specifications.*

***Protocol ownership** - when the IETF takes ownership of a protocol or function, it accepts the responsibility for all aspects of the protocol, even though some aspects may rarely or never be seen on the Internet. Conversely, when the IETF is not responsible for a protocol or function, it does not attempt to exert control over it, even though it may at times touch or affect the Internet.*

On the *multi-stakeholder governance* of the Internet the United Nation has produced an extended report. (MacLean 2004).

2 – The World Wide Web

Competitive factors

Hypertext origin goes quite far in time: "oNLine System" was made by Doug Englebart[6] in the 1970s, and Ted Nelson's Xanadu planned network access to documents[7]. Another hypertext system called *Gopher*[8], was used since 1990 as a document sharing system, years before the success of the *world wide web*. Unlike graphic browsers, Gopher could be used from simple ASCII terminals, but did not allow the simultaneous presence of text, pictures, sound in the same document. In addition to this technical reason, also Gopher's licensing status played a major role in its gradual decadence.

[...] Mosaic could match the functions of the Gopher protocol and additionally offer added functions such as hyper linking from within HTML files which brought together related

6 See video recordings: http://sloan.stanford.edu/mousesite/1968Demo.html

7 See http://xanadu. Ted Nelson eventually released Xanadu source code under the name Udanax in 1998: http://www.udanax.com/

8 On Gopher, see the *gopher manifesto*: http://www.scn.org/~bkarger/gopher-manifesto
Some Gopher server survived and can be reached with recent browsers. As an example, see: gopher://gopher.quux.org/1/Software/Gopher

pages more efficiently than Gopher, there was no longer a compelling reason to choose the Gopher system. Another advantage the early Web had over Gopher was the decision of the University of Minnesota not to definitively rule out the option of exercising its intellectual property rights over the Gopher protocol, for any other organisation deciding whether to devote time, effort, and expense to adopting one of the systems the possibility of getting locked into a technology that they could then find themselves being charged for was good reason to prefer the World Wide Web. Most of the files and databases that had been available on Gopher were converted into HTTP compatible formats and made available on the Web [...][9]

The growing information availability still drives the expansion of the Web, which in turn has become the leading factor for *Internet* demand and expansion.

Layering and modularity

Basically the web is a client-server architecture based on two layers: the first is the language that prescribes the way pages are composed (HTML - *hyper text markup language*) and the other regulates information flow between web servers and client browsers. In this second layer the most used protocol is HTTP (*hyper text transfer protocol*), but many others are used for different functions (as FTP for file transfer): all those protocols can be seen as modules. Both layers are becoming more and more complex as the Web evolves and more protocols are added in a cumulative process (Tuomi, 2002).

Search engines can be seen as a further layer, that makes it possible to access information through its *content* and not only through an *address* that should be known in advance.

The thing called "Web 2.0", even being quite elusive, is for sure a *platform*, a layer upon which modules (and other layers) can be built. The compact definition given by Tim O'Reilly (2005) says:

Web 2.0 is the network as platform, spanning all connected devices; Web 2.0 applications are those that make the most of the intrinsic advantages of that platform: delivering software as a continually-updated service that gets better the more people use it, consuming and remixing data from multiple sources, including individual users, while providing their own data and services in a form that allows remixing by others, creating network effects through an "architecture of participation," and going beyond the page metaphor of Web 1.0 to deliver rich user experiences.

As said for the Internet, web modularity can be found in its *technologies* (different protocols for different functions) but also in its *resources:* independent web sites form networks linking their information together with the same coalescence process seen the Internet.

9 See *Codeghost*: http://www.codeghost.com/gopher_history.html

Cooperation, coordination and governance

The cooperation enabled by web resources (such as wikis) can be seen as another form of information modularity. Benkler (2006) deeply analyses this process, which he calls *peer-production*. The Web offers an environment, a context, in which information becomes accessible, usable and can become knowledge. As said, a major role in making information usable is performed by search engines[10] which sit on top of the complex layering of sites and try to extract meaningful information. In the future, semantic web and folksonomies, and in general projects based on users tagging information[11] in a cooperative way will assume a growing importance, complementing search engines.

Principles and strategies

Web governance consists in defining technical standards: this is the goal of the international World Wide Web Consortium (W3C). Even if open to anyone, W3C activities are aimed mainly at organizations and firms. W3C recommendations aren't mandatory standards, and are royalty-free. The declared *mission* of W3C is:

> *To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web.*[12]

The governance process requires the maximum attention to any stakeholder:

> *All stakeholders can have a voice in the development of W3C standards, including Members large and small, as well as the public. W3C processes promote fairness, responsiveness, and progress: all facets of the W3C mission.*[13]

Developing an open Web explicitly designed for sharing knowledge is one of W3C's goals (Bratt 2006):

> *The social value of the Web is that it enables human communication, commerce, and opportunities to share knowledge. One of W3C's primary goals is to make these benefits available to all people, whatever their hardware, software, network infrastructure, native language, culture, geographical location, or physical or mental ability.*

3 – The Linux Kernel

As already said, the Gnu/Linux operating system has many components: in addition to the Linux kernel there is a collection of system programs developed by the GNU project, and many other programs from different other projects. Therefore different GNU/Linux *distributions* reflect different ways to pick, merge and blend those different elements. The

10 Google holds a position of substantial monopoly in the search engines marketplace. The importance of the process of information selection and the lack of transparency of Google's ranking algorithms raises concern.

11 See for instance http://www.del.icio.us

12 http://www.w3.org/Consortium

13 http://www.w3.org/Consortium/process.html

process of building a distribution is the object of part three, while this section is focused on the Linux *kernel*.

Competitive factors

Linux is a product among others in a segmented operating system market. As its main competitor MS Windows, has explicitly developed specific strategies to compete in the different segments of the market. The subject in charge to work out those strategies is The Linux Foundation[14], who has members from the whole ITC industry and "*promotes, protects and standardizes Linux by providing unified resources and services needed for open source to successfully compete with closed platforms*".

Even if Linux is quite weak in the desktop system vendors segment, where computers are sold with Windows already installed[15], servers can be bought also with one of the GNU/Linux most popular commercial distributions. Reportedly, GNU/Linux server installations are growing more than its competitor (Fondati 2006, Shackland 2004).

In the *equipment suppliers* segment (appliances, industrial control systems, phones, and other embedded systems) Linux competes with, among others, VxWorks, QNX, and Microsoft (which has a share similar to others). Equipment suppliers often install different operating systems for different products, so they don't seem to have a precise direction. However interest for Linux appears to be growing (Turtley 2005).

Different *Linux OS providers* are competing between them *and* against other proprietary competitors: some are given for free (as Debian), while other are commercial (RedHat and Novell are the best known). Many offer a product (a specific GNU/Linux distribution) and maintenance services, while several firms offer maintenance alone.

It has been observed that once free/open source software enters a market, it is very difficult to find competitive strategies to expel it, as users tend to stick to it fearing proprietary lock-in (Lindman 2004).

Layering and modularity

Operating systems are, as general rule, layered. Moreover, the core (kernel) is a layer on which other elements of the system are based, and is modular in most operating systems, both for technical reasons and to ease its development. As in networks, modularity appears at two levels: the kernel as a *product* is modular, as the *process* to build it. The second part of this paper will study this aspect in greater detail.

14 *"Founded in 2007 by the merger of the Open Source Development Labs and the Free Standards Group, it sponsors the work of Linux creator Linus Torvalds and is supported by leading Linux and open source companies and developers from around the world":* http://www.linux-foundation.org/en/Main_Page

15 On having GNU/Linux installed by PC vendors see Vaughan-Nichols 2007

Cooperation, coordination and governance

Linus Torvalds, first author of Linux, still coordinates and has the "last word" on decisions regarding the project, which is still Internet- and Web-based. But Linux has also become a huge business: since year 2000 the firms most involved in the project have founded a no-profit corporation, the Open Source Development Laboratories (OSDL), now part of The Linux Foundation. Those labs are located in USA and Japan, and work to accelerate the adoption of Linux. While the development continues in the community, the Foundation has the purpose of *promoting, protecting, and standardizing Linux and open source software*. Even individuals can register as affiliates for a modest fee, but no member has any right on property or assets of the corporation.

Principles and strategies

Such a large coordination between firms and a large community of users and developers around a single project under a common strategy is unprecedented and raises some interesting issues.

OSDL declared *mission*[16] is:

> *To accelerate the deployment of Linux for enterprise computing through:*
> *- Enterprise-class testing and other technical support for the Linux development community.*
> *- Marshalling of Linux-industry resources to focus investment on areas of greatest need thereby eliminating inhibitors to growth.*
> *- Practical guidance to our members - vendors and end users alike - on working effectively with the Linux development community.*

These are essentially strategic goals: identify and eliminate threats in the market and weakness points in product, develop efficient production processes, exploit opportunities. Members of OSDL are often competing firms, nonetheless they contribute developing common strategies around a product that, staying free, allows them to develop competitive advantages. Why firms competing between them should invest in a innovation project they don't even *own*? In *Corporate Philanthropy* (Porter, Kramer 2002), this kind of action is called the *construction of the competitive context*. Authors identify four fronts:

> *A company's competitive context consists of four interrelated elements of the local business environment that shape potential productivity: factor conditions, or the available inputs of production; demand conditions; the context for strategy and rivalry; and related and supporting industry.*

In detail, those four fronts are present in the building of the competitive context for Linux.

16 http://www.osdl.org, now http://old.linux-foundation.org/about_osdl

1. *Factor conditions.* One single product, Linux, is a common infrastructure technology available for each actor, in a context where the only conditions needed for production are net access, available to anyone at the same conditions, and the ability to innovate.
2. *Context for Strategy and Rivalry.* The competitive context is being defined through a common policy, set by the GPL licence, which is valid for each stakeholder: programmers, firms, users, all are in the same conditions with respect to code.
3. *Related and Supporting Industries.* A standard product allows firms to work in similar conditions on the same product, leveraging their strategies and creativity to compete.
4. *Demand conditions.* Linux *sophisticated users* not only are able to anticipate the demands of most users, but can enter in the production cycle adapting the product to their needs and submitting their proposals to the project.

The following statement seems suitable (Porter, Kramer 2002) to describe what happens around free software:

> *Philanthropy can often be the most cost-effective way for a company to improve its competitive context, enabling companies to leverage the efforts and infrastructure of nonprofits and other institutions.*

Another point of view, focused on innovation rather than the competitive context, could come from *Open Innovation* (Chesbrough, 2003). Even if never abandoning a "traditional" view over property of knowledge, Chesbrough pictures effectively the extreme difficulty and definitive obsolescence of the internal (closed) innovation paradigm, due to erosion factors that *have loosened linkage between research and development*.

> *Ideas can no longer be inventoried on the shelf, because they will leak out to the broader environment over time. A company that fails to utilize its technology may later see variants of those ideas exploited by other firms. At the same time, these erosion factors collectively create rich variety of possible research inputs available outside the firm.* (Ch.2)

The solution is to let firms be open to a constant flow of knowledge (both in input and output) with the environment.

> *Companies must structure themselves to leverage the distributed landscape of knowledge, instead of ignoring it in the pursuit od their own internal research agendas.*

Chesbrough also recommends that firms develop a suitable business model to *claim a sufficient portion from the [value] chain to justify its participation*.

4 – The Debian project

Installing one of the tens of thousands different free/open source programs whose source code is available around the net may be complicated even for a skilled system manager. *Distributions* are collections of programs already compiled in *packages*, ready to be installed, designed to reduce the hassle of compiling the source code.

The Debian project's work is to produce and maintain the widest and most complete GNU/Linux distributions, which contain more than 15'000 different programs: it is probably the largest operating system ever realized (Amor-Iglesias *et al,* 2005). Debian , as explained in part three, embeds in the distribution the knowledge from its community of volunteer experts.

Competitive factors

Linux system providers offer different distributions in competition between them. Some of them use the software packages from Debian distributions (for example Linspire), while other use their own criteria, as RedHat and Novell. Distributions are differenced on their target users, the variety of programs they offer, the set of services and their price[17].

Layering and modularity

In a system installed from a distribution, the single package can be seen as a module. Designing good interfaces between those modules is the key for a stable system and a successful distribution. Layering in a Debian system emerges from the interdependence between packages: some of them require others to be installed, in fact imposing a hierarchy of layers (the base packages at the bottom, user programs at the top).

Cooperation, coordination and governance

Based on the *Debian constitution*[18], Debian has a complex division of tasks between technical roles (*developer*), hired with a severe selection process, and coordination roles (*project leaders*), being elected by developers. Debugging, writing documentation and user support activities are based on the wider community of Debian users. There is also an elected *leader* of the whole project with mainly coordination and communication functions.

The Debian project's property is owned by *Software in the Public Interest inc*[19] (SPI), which also provides legal support.

Principles and strategies

There are a number of documents describing Debian's goals and strategy. The *manifesto* states that the whole work process is centered on the community's needs (*The Debian Linux Manifesto*[20]):

The Debian design process is open to ensure that the system is of the highest quality and

17 For a review of availilable distributions see: http://en.wikipedia.org/wiki/Comparison_of_Linux_distributions

18 http://www.debian.org/devel/constitution

19 http://www.spi-inc.org

20 http://www.debian.org/devel/join/newmaint

that it reflects the needs of the user community.

The *Debian Social Contract*[21], along with the *Debian Free Software Guidelines,* define the project's traits and what features its products should have, outlining a long term *strategy*. Point four of the *social contract* is very clear on how the project mission should create an open *environment* for users.

> *4. Our priorities are our users and free software*
> *We will be guided by the needs of our users and the free software community. We will place their interests first in our priorities. We will support the needs of our users for operation in many different kinds of computing environments. We will not object to non-free works that are intended to be used on Debian systems, or attempt to charge a fee to people who create or use such works. We will allow others to create distributions containing both the Debian system and other works, without any fee from us. In furtherance of these goals, we will provide an integrated system of high-quality materials with no legal restrictions that would prevent such uses of the system.*

Common elements

The analysis of these four projects shows interesting common elements. Each project is an *enabling environment* in which knowledge is a key factor. Moreover all the projects can be seen as a whole, each being a layer of a overall entity, which contributes to build a new environment whose value is greater than the sum of its parts.

These are the common elements in synthesis:

1. Layered and modular **structure.** Knowledge plays a major role in the division between horizontal layers and vertical modules:
   - each level or layer needs a different kind of knowledge or competence. At a given layer, the knowledge needed at another layer can be ignored;
   - different modules inside a layer address different problems or projects, but require the same “set of knowledge”;
   - layers and modules relate to each other through well defined, standardized interfaces and protocols, that address and regulate every aspect of the dialogue between them;
   - process cooperation and formalized interaction between modules becomes possible on the basis of those standards.
2. **Cooperative** model. All subjects developing and taking care of the system modules cooperate using predefined interfaces and tools. Coordination of the cooperative

21 http://www.debian.org/social_contract

process (*governance*) can happen on loosely hierarchical basis, with more or less democratic procedures, but in general based upon rough consensus. Product modularity is reflected in a cooperative, and modular, *production* process.

3. **Openness** on all levels:
    1. *use and distribution*: no limitation for use of the product or project. Anyone can use it, without barriers on purpose or intent;
    2. *product architecture*: no barrier of technical, normative, economic nature is opposed to the access to information that describes and defines the project. Anyone can suggest and try to implement new elements or modules, or modify and improve system architecture or process;
    3. *governance*: project management and decision process are potentially open to any stakeholder. Nearly anyone with enough technical competence and necessary consensus can access to the coordinating roles.
4. **Shared principles.** All stakeholders generally share some general principle: a mission, guidelines, ethical code or some other foundation document. The more the participants become involved in the project, the more they account those principles as important. The foundation documents include not only some ideal principle, but also a *vision,* operative guidelines and future directions. These respond to stimuli (threats and opportunities) coming from the environment, and consequently are (at least part of) a strategy. Those principles give an order (in both meanings: *command* and *regular arrangement)* to a structure that otherwise leaves to individuals and groups the largest autonomy inside the single modules. This looks like a way by which letting chaos produce the maximum creative yield without getting outside of the expected objectives.

The building of an *enabling environment*.

An "*enabling context*" is defined as (Von Krogh, Ichijo, Nonaka 2000) "*what drives knowledge creation*" and is linked to the concept of *Ba* as an organization's shared space that facilitates interaction, exchange and sharing of knowledge and creativity. The examples examined in this paper cannot be seen as a space that is *internal* to something. Nobody *owns* them. Instead of being an environment where actors meet to build some other thing, they are an "external" context that actors contribute to build. A free software project as Linux, for instance, cannot be imagined as being inside something, not even its own community (while the contrary can be said). It can be seen as if it is floating in the net, where it grows and contributes providing its own environment.

Building, enhancing and empowering the environment is the *objective* of the action but also – simultaneously – the *means* by which the activity takes place. The goal is building an environment that is never completed, and is incessantly manipulated, with

new modules and layers added. The environment also is a production factor of itself, being made of spaces, relations, infrastructures, code, projects, informations. Many firms are learning to take advantage of those environments as a place where innovation happens (Chesbrough 2003) and where to contribute with innovation and investments.
An open environment is effectively *enabling* towards *any* actor: anyone can become involved enhancing the environment and contribute to its development, and all users of the environment can potentially be co-workers. Alongside of cooperation, also *competition* is part of the environment, providing not only the stimulus and energy necessary for evolution, but also the conditions of selection by which only the best solutions are widely adopted and evolve.

As a synthesis of this rather fuzzy description, an enabling environment can be seen as a dynamic context with some characteristics (full openness, modularity, layering) that allows an high degree of interaction between actors that do not own the environment but contribute to build it.

II – Internet and strategy in the competitive context of software

The Internet is not only opening up entirely new categories of applications, but it is irrevocably changing the ways in which software is sold and involving software technologies in contentious debates over many public policy issues including national sovereignty, national security and law enforcement, and limitations to information access.
Messershmitt, Szypersky – Software Ecosystem

Get the strategy and the management side right, and the software business can be like having a license to print money [...] but get the business model wrong, and [...] software companies can resemble dinosaurs futilely trying to escape the death grip of an ancient tar pit.
Michael Cusumano – The business of Software

Software market: economies and life-cycle

Software creation is a well-articulated process in which some components can be industrialized, while others are still crafted “by hand”, depending on specific competences. Needing constant innovation, software production depends on creativity.

As noted by Brooks (1995), the production of software not always follows economies of scale, especially for the aspects of debugging and testing.

The bearing of a child takes nine months, no matter how many women are assigned. Many software tasks have this characteristic because of the sequential nature of debugging.

Instead, economies of learning have great relevance, since they give the ability to manage reliably both technology and the complex production/testing process. Messerschmitt and Szyperski (2003) list the steps in the software life-cycle where knowledge is transformed in a product:

The major steps in a life-cycle include analysis of user needs, development of the software (including architecture design, programming, testing, integration, and other functions), provisioning of the equipment and software in the user environment (including user training and often organizational changes), operation and administration of the software, and use. These steps form a value chain, where each step depends upon the successful completion of previous steps and adds value to them. The entire cycle is typically repeated multiple times, and the software supplier faces interesting challenges in managing the process of maintenance and upgrade of the coexisting version of the software while it is in use.

This value chain is the same for proprietary and free/open source software, but the latter has some peculiar way to find its resources, relying on the involvement of a community of users-developers (Senyard, Michlmayer 2004):

We describe a three phase life-cycle for free software projects. The first phase is characterised by ***closed development*** *performed by a small group or developer with much in common with traditional software development from which we have named the cathedral phase in reference to Eric Raymond. The second phase is a move from traditional development to* ***community based developmen****t which we have named the transition phase.*
Only projects with certain properties can successfully pass the transition phase. In order to operate successfully in the bazaar phase a number of activities must be completed. [...]
The final phase is where the project becomes a community based project and gains the associated advantages -- we have named this the ***bazaar phase****. The life-cycle model proposed here gives a better understanding of the dynamics of free software and can assist in their success.*

The third phase, more critical but also fertile, is where a free project differentiates the most from a proprietary one, to the point it tends to behave like a movement (Neff, Stark 2002):

Open source projects display a responsiveness that has allowed users of products to be more directly involved in the design and the design-in-use of these products. Open source

> *organizational forms challenge the traditional notion of boundaries between for-profit and non-profit organizations, as they often have more in common with social movements than traditional formal organizations.*

The role of users-developers is a key factor in one of the most critical phases of the life-cycle: distributed debugging. It is very important to notice that proprietary software producers cannot take advantage of this factor, for which the availability of source code to a large base of expert users is a prerequisite.

In an open environment, free/open source has a competitive advantage over proprietary software thanks to debugging. This is the most known advantage, but not the only one.

Internet, value chain and strategies

Michael Porter observed in 2001 that Internet would have a role in shaping the value chain and making the best strategies emerge (Porter 2001):

> *The great paradox of the Internet is that its very benefits - making information widely available; reducing the difficulty of purchasing, marketing, and distribution; allowing buyers and sellers to find and transact business with one another more easily - also make it more difficult for companies to capture those benefits as profits.*

and also

> *Many have argued that the Internet renders strategy obsolete. In reality, the opposite is true. Because the Internet tends to weaken industry profitability without providing proprietary operational advantages, it is more important than ever for companies to distinguish themselves through strategy. The winners will be those that view the Internet as a complement to, not a cannibal of, traditional ways of competing.*

Those benefits have been specially effective for free software all along the value chain. This section will use the elements of the well known framework identified by Porter, and highlight the impact of Internet in the software value chain.

On the ***inbound logistics*** activity, Internet has provided a "place" where helpful programmers meet free software projects, promoting the encounter of demand and offer. The difference between free software and proprietary projects is that the first can "hire" a programmer *after* he has contributed to the project, and can reject bad work, while the latter has to face a cost for selection and hiring a programmer in advance. Benkler (2002) highlighted the *information gains* of the process he calls *self-identification*, but also the disadvantages and their costs.

This common workplace makes easier for who has skills and knowledge to use and improve them. The motivations for writing free software are complex, and a literature survey and original hypothesis can be found in Rullani (2006). One prominent motivation is to improve one's knowledge and signaling of personal skills through the public availability of source code. These two are possible only with free software and an open environment as the Net.

Also ***development*** of software take advantage of the Internet. In the case of free software, instead of being located inside the firm, the code is *into the environment*: development activity can take place from anywhere, since organization and coordination factors are provided by the environment. Of course also proprietary software firms can have their software written from everywhere in the Net, but they have to *extend* the firm environment and actively provide the coordination factors where development takes place. This difference is well pointed out by Benkler (2006):

> *The core technologically contingent fact that enables social relations to become a salient modality of production in the networked information economy is that all the inputs necessary to effective productive activity are under the control of individual users. Human creativity, wisdom, and life experience are all possessed uniquely by individuals.*

This opens a new possible interaction between traditional (market-based) production and social-based *non-market* production. Benkler (2002) also suggests that commons-based peer production has advantages in improved identification and allocation of human creativity.

> *We would say that when the cost of organizing an activity on a peered basis is lower than the cost of using the market or hierarchical organization, then peer production will emerge.*

On the ***outbound logistics*** activity, The Net has favored the use of free software much more than proprietary software, given the restrictions the latter imposes on copy and replication. These barriers binds proprietary software to traditional distribution channels through *distributors* and *value added resellers* in a hierarchical and unidirectional way (Wallace 2006). For proprietary software, easy copying is a threat and a weakness, and imposes costly, unpopular and often ineffective copy protection systems. On the contrary easy copying is a strength and opportunity for free software: immediate gratification, reduced network effect startup costs, easier maintenance and upgrade (Messershmitt, Szyperski 2003).

This introduces us to the advantages in the ***service*** activity of the value chain: easily upgrading programs, especially for security reasons, is an absolute requirement.

Oracle's *chief security officer* comments how critical is delivering software updates to customers (Davidson 2005):

> *Vendors may also need to provide fixes on multiple versions/platforms or bundle multiple security fixes together to minimize patching costs to customers, not to mention various*

> *testing on the products shipped to ensure the fix does not break anything else.*
> *As an example, Oracle has done 78 fixes for a single vulnerability, which took more than five days to complete. We also release bundled fixes quarterly on dates tied to financial reporting calendars (e.g., many customers will not touch their production systems during quarter-end). A two-line code change can take five minutes, but getting a fix into customers' hands in such a way that they will apply it takes way more than a few minutes.*

In the same article, Davidson explains how critical can be the relation between proprietary software firms and an active and participating community regarding security issues.

In conclusion, there seems to be a clear advantage for free software in a software production context more and more centered around Internet. This happens all along the value chain, but specially in the outbound and inbound logistics.

III – Environment and knowledge value chain

> *[...] the only way to protect ideas is to not communicate them, which in turn dramatically reduces their intrinsic value.*
> Sawhney, Prandelli – *Communities of Creation*

Modularity allows the reduction of complexity and helps the eventual emergence of layers. In change, an effort in the definition of interfaces between modules is needed for them to interoperate. This aspect is critical for three reasons: first, interface definition is the point where *production* activity inside the single modules needs *coordination* at the project level, or seeing it the other way, coordination decides what modules should do. Second, while code can be corrected, interfaces – once defined – are rather immutable and introduce an irreversible element. Third, interfaces are critical for building the overall environment: if interfaces between new modules and layers and the ones present in the environment are compatible, new elements add nicely to existing ones in a *cumulative* way, layer after layer. This happened when the Web was added to the "bare" Internet, and is happening with Web 2.0. The environment, once one layer is functional, enables the creation of successive layers.

Making things work on such a scale means *embedding knowledge* about the functions of layers and modules into interfaces and protocols. Interfaces define knowledge boundaries, and drawing where interfaces should be means identifying *knowledge domains.* What lies inside is the knowledge pertaining to actions performed by the module, the rest must come from the environment. Too many interfaces or too few, and the project gets buried in complexity or becomes unmanageable. Protocols instead define how *action* between elements should take place. Once fixed in interfaces and protocols, knowledge is frozen into the environment and sticks more or less permanently to it. Standards emerge when some knowledge has to be reused, making a formal description useful. The number and position of interfaces is what determines what Benkler (2002, 2006) calls the *granularity* of project modules.

The way knowledge cumulates and interfaces are laid between modules and layers is important: this process gives the project and the environment its shape over time. The next section will try to explain how this happens in the Debian project, following a knowledge value chain (Lee, Yang 2000) from the programmer to system user. But first, a few words must be spent on what modularity means in the software industry.

Modularity and components standardization

It has been observed how modularity helps in managing innovation and differentiation but at the expense of coordination (Devetag, Zaninotto 2001):

> *Modularity seems better suitable than other approaches when the adaptation to the demand variety and the speed of search in a given problem space (i.e., the rate of technological innovation) are more important than tight co-ordination. If this is the case, division of labour must occur in such a way that considerable space for local adjustment and experimentation is assured, unlike the case of a traditional assembly line.*

In the Linux *kernel*, as in other projects, modularity is a tool to minimize complexity generated by the interdependence of components. The goal for modularizing is (Narduzzo, Rossi 2005):

- manage uncertainty and variability in problem solving,
- ease task and product decomposition,
- widen product variety, on a differentiation strategy.

Given the extreme division of tasks that a free/open source project is likely to face, modularity is not only a technical need, but a development model (Narduzzo, Rossi 2005).

> *We argue that modularity, which can be regarded as an innovative manufacturing paradigm for the design and the production of complex artifacts, is a key element in explaining the development and the success of many F/OSS projects, and it offers a comprehensive explanation of many key issues in F/OSS development, such as how division of labor takes place within developers, how coordination is achieved and how code testing and integration is deployed, how innovation occurs, and so on.*

Modularization sometimes fail: knowledge embedded in interfaces must be carefully planned in advance, and – as system evolves – the boundaries between its modules must fit tightly to each other (Narduzzo, Rossi 2005).

> *[...] Unfortunately, this neat description of modular design sometimes does not succeed; most of the times, after the integration of the independently developed modules,*

inconsistencies come up on and the system does not work properly. The most common reason for this failure is the emergence of some interdependencies which were left out at the beginning, at the time of architecture and interfaces definition.

This necessary careful planning of the boundaries between independent knowledge and action domains takes the name of *architecture* (Messershmitt, Szyperski 2003):

Software architecture is the first stage of software implementation. It decomposes and partitions a software system to modules that can be dealt with somewhat independently. This modularity has significant business implications, including open interfaces and API's, system integration, and the composition of different applications and infrastructure.

For efficiency reasons the software industry – not only free/open source – often develops its products using reusable and standard components (Messershmitt, Szyperski 2003).

There is growing interest in software reuse and component software as a way of improving the productivity of development organizations, addressing increasing complexity, and improving quality. The idea is to assemble applications largely from existing components, which are modules created independently of any particular system context and constructed for multiple uses.

Reusing means that large number of interdependent modules must interact. This implies defining common interfaces and standard protocols based on shared knowledge. Openness gives the possibility to new layers to interface with previous ones and form an open, enabling environment. As Messershmitt and Szyperski (2003) explain:

Coordination among firms is essential to getting operational software into the hands of the users. Standardization is one way in which software suppliers coordinate themselves, and it is growing in importance as distributed applications move across administrative and organizational boundaries. Driven largely by the rapid success of the Internet and distributed applications and new requirements for mixing different information media within a single application, the industry organization has been moving from a vertical to a horizontal structure, the latter consonant with an architecture concept called layering. Layering provides a relatively flexible way to partition an infrastructure to provide a wealth of services, with the possibility of adding additional features and capabilities by building layers on top of an existing infrastructure.

This is the kind of environment where software, both open and proprietary, is now developed. But, as already shown, free software benefits, at least in many cases, from competitive advantages.

In this specific context the goal of Debian is to take free software as produced from programmers, and provide the knowledge that is necessary to make it interoperable and available for anyone who wants to build a system, or a distribution, or whatever. This is the closest thing to an open and standardized software components “market” ever made.

Knowledge value chain: the GNU/Linux distributions example

How does an open source program become usable? What are the steps from production to use, and what actors are involved? What content of knowledge is being added at each step? Roughly, the value chain can be summarized as follows: *code* written by a programmer becomes a part of a *program,* which is compiled and enclosed in a *package* that is part of a *distribution.* When used, the package takes its place as a module in a running *system.* (see Table 1: Value chain from code to system).

phase	who	what	scope	adds
creativity	programmer(s)	code	project developers	analysis, talent
life support	users	program	community	debugging, documentation, websites, mailing lists, repository
stability	package developers	package	distribution users	compilation, configuration, dependencies, documentation, tools
completeness	distribution developers	distribution	generic users	repository, tools, support, documentation, security
functionality	system manager	system	local users	local configuration, documentation

Table 1: Value chain from code to system

Code

A programmer (or a group) writes code after a problem analysis, following talent and experience. She can write new code and/or reuse (even modifying) existing one. Libraries of existing code can be used, and since different libraries can do the same job, she has to choose the best one. Programmers should also know if they are violating licenses or patents, and if the license scheme for their new program can be combined with those of the libraries they link to. This is the kind of knowledge embedded in this phase: analysis and *creativity*. Once the program is written and even debugged, it has to be maintained, and hopefully upgraded: so, several different versions can be simultaneously on the bench. Since different versions can make use of different libraries, dependencies issues can arise. The goal of this phase is getting code working, often not even distributed: its scope is the group of developers that are working on the program.

Program

Before code can be used by its targeted users, several other tasks must be completed: it should be integrated with one or more different operating systems. If the systems on which the program has to be installed isn't the one used for development, *porting* is needed. Moreover, documentation should be written, websites, mailing lists for help, news and debugging coordination provided. In this phase the supporting community of users is involved in providing *life support* for the program through debugging, porting, asking for new features and upgrades, writing documentation, help, and so on. The kind of knowledge embedded in this phase regards interaction with users and environment. The scope is the community.

Package

Usually the program is distributed in source code. As already said, installing it requires resolving all needed dependencies and can be a long and frustrating process, especially for big and complex programs. To address this problem, distributions can manage packages which contain the pre-compiled program, documentation, configuration files and all information needed for integrating it into the given system: description of the program, its dependencies on other programs, libraries or modules, alternatives, incompatibilities and so on. The added value with respect to the lonely source program is enormous, and allows saving huge amounts of time to system managers. The knowledge a package includes is the experience of a skilled system manager who knows very well the system architecture, so that the coherence of the system is maintained. It gives both the program and the system the *stability* needed for reliable use. Who carries this task out? In the case of Debian Project's distributions, the group of Debian Developers. The scope of the package are all users for a given system: for instance users of a Debian distribution or providers of Debian-based distributions.

Distribution

A distribution is a coherent set of packages, plus tools, documentation and support for their installation and maintenance in a complete, stable and secure system. Each distribution responds to some specific need: some are focused on general purpose systems, as Debian, Redhat, Novell, Ubuntu; other on specific tasks as composing and playing music, security system analysis, and so on. The goal is to provide *completeness* with regards to the intended use. Added knowledge comes from the overall functionality, as choosing the base set of packages for the system and the other packages specific for the expected prevailing use, or the mechanisms to keep the system stable and secure, which often require online repositories for upgrades. The scope is all potential users within the given distribution's goals.

System

At the system level, system managers (or users which manage their own system) can choose which packages to use within a given distribution to match their specific needs. The main goal here is obtaining a *functioning* system. This means completing configuration steps to let the system connect to its network, authenticating the right users, performing specific tasks as dispatching e-mails or playing music. The knowledge the system manager is embedding in the system at this point has a local nature such as IP network numbers, passwords, user names, web server configuration. Even the scope is local: only users of that system are involved in the choices of their system manager, who in turn has no need to know the intricacies of the layers beneath (except in case of troubles). System manager doesn't need to bother about dependencies between packages in a distribution, nor incompatible libraries used by different programs, nor licensing issues behind such and such chunk of code. Those are deeply sunk and frozen into the underlying knowledge structure.
This, said, in case he has troubles, he can always refer to the communities of the layers where the problem arises: primarily the program and distribution layers. Apart from solving the problem, this dialogue has two effects: first, people exposing a problem becomes (even temporarily) part of the community, sharing values (information, trust) and providing feedback on the product; moreover, the dialogue with the community becomes part of the embedded knowledge in the resources of the *life support* and *distribution* layers in the form of mailing lists archives, FAQ pages, etc.
This section tried to give an example of how knowledge embedding plays a major role in building an open, enabling environment. The same arguments used here for a GNU/Linux system may be applied to the Net as a whole, considering how its evolution is based on successive layers that grow on underlying ones embedding cumulatively additional knowledge. Given the nature of the process and the means used, most of the knowledge remains explicit and readable.

The environment as a learning system

Involving new subjects is an important point that emerges examining both the distribution knowledge value chain and the software supply chain (in part two). Recruiting new developers, leveraging on the community of users, letting the project interact with other projects, is a major role played by *knowledge brokers* (Sowe, Stamelos, Angelis 2005):

> *Knowledge brokers not only help individuals manage and extract valued software knowledge from software repositories, but also help OSS projects to engage in a discourse and co-learning experience with their user communities.*

This socialization process lays the community foundation and gives a hint on how the project as a knowledge base guarantees its reproduction (Sowe, Karoulis, Stamelos 2005):

*The F/OSSD context is a "*symbiotic cognitive system*", where the community learns from its participants, and each individual learns from the community. Ongoing interactions and activities are a means of acquiring valuable knowledge that is worth archiving. Participants learn advanced and basic concepts associated with collaborative software development. Novice and lurkers alike learn by browsing the knowledge base. The legitimate peripheral participant learns as an apprentice, observing and participating. [...] Thus, as a bricoleur, the learner is continuously involved in a discovery, trying to put order where there is disorder, structure where there is no structure, fitting his or her knowledge with the project's knowledge base.*

It's very important to consider that this learning dialogue is not volatile, but stays available in the form of emails in mailing lists, irc logs, FAQ's in website pages, documentation, foundation documents, requests for comments, etc. The dialogue so crystallizes and becomes explicit embedded knowledge; it sticks permanently to the environment, becomes indexed by search engines and recoverable from a content search. This *delayed dialogue* makes possible to knowledge to cross the boundaries of the community where it happened and become part of the assets of the project.

Knowledge brokers can stir knowledge between communities, and play an important role in innovation. Ilkka Tuomi (2002, ch.7), starts from the concept of layered *ba*, defined as a *"field of meaning creation where innovative artefacts and their uses become mutually constructed"*, and subsequently makes a difference between two different types of *ba*:

When the ba relies on resources that are embedded in the community that provides the foundation for the ba, we can talk about an evolutionary ba. When the ba relies on on resources that are not embedded in the focal community, we can talk about combinatory ba.

Tuomi then considers then how innovation diffuses between the two types of ba.

Innovations [...] also have two qualitatively different paths of diffusion. New knowledge can cross the boundary between ongoing interaction and institutionalized social practice, and sediment in the institutions of the focal community. Innovations, however, can also diffuse by inducing change in external communities. Such inductions across communities is often mediated by artefacts. We could simply call the former vertical diffusion and the latter horizontal diffusion.

The projects considered in this work, along with many others, have an extraordinary effective innovative value: not only they produce valuable artifacts, but doing so they build enabling environments for further development. This dynamical process brings a consequence: a choice (such as an interface, a protocol or a license) made at some point becomes irreversibly embedded when other layers are built upon it, and subsequently direct or polarise further innovation in a specific direction, tracing a path for future evolution.

Conclusions

The Tao gives birth to all beings,
nourishes them, maintains them,
cares for them, comforts them,
protects them,
takes them back to itself,
creating without possessing,
acting without expecting,
guiding without interfering.
That is why love of the Tao
is in the very nature of things.
Lao Tsu – *Tao Te Ching*[22], *51*

Synthesis between enabling environment and knowledge value chain

The first part of this work analysed some of the distinctive traits in four of the projects that are at the base of the present information revolution. Some aspects are here considered fundamental: layering, modularity, openness both at the product, process and governance levels. All of these projects, even if they don't have a strong central coordination, are developing strong identity and strategy: they sometimes behave like firms and compete successfully with them. In the case of Internet and Web, noncommercial projects pulled commercial competitors completely out of business. Those kind of projects, thanks to the key factors explained above, flank each other and sometimes combine in layers forming environments that enable others projects to grow upon them. Sometimes positive feedbacks enhance this enabling effect: free software and hypertexts existed before the Internet, but Linux and the Web owe their success to the diffusion of Internet. Internet demand, in turn, exploded thanks to the diffusion of the Web.

The way those environments are enabling each other has been the argument of part two, in particular considering – in the market of software – the role of Internet and the competitive advantages it gives to free software.

The mechanisms of layering of knowledge is at the basis of the environment's effectiveness. The knowledge value chain of a GNU/Linux distribution has been analysed (part three) in some detail, highlighting how the output of one layer becomes the enabling factor for the next, incrementally embedding knowledge. The way knowledge builds up

22 English translation by S. Mitchell, from: http://acc6.its.brooklyn.cuny.edu/~phalsall/texts/taote-v3.html#51

and crystallizes in the environment enables innovations to easily cross boundaries of projects and communities, and allowing the creation of new ones.

Firms and innovation in the context of enabling environments

Internet and the Web are so large projects that hardly any centrally planned commercial enterprise could have built them alone; but they have enabled the possibility for many firms to exist, from telecommunication companies to ISPs, search engines and online shops. Furthermore, Internet would not be possible, nowadays, without commercial telcos and ISPs.

Linux is at the center of huge investments from many ITC firms that team up to build their competitive arena in the future, around a project that is both an innovation and standardization opportunity. The Debian project is the closest thing to what can be called a software component provider for the operating system environment (including the operating system *market*), and competes successfully with firms as Novell and RedHat. Moreover, even being a no profit project, Debian provides its work to commercial firms.

Firms participating to open projects and fostering the growth of new environments are coherent with both *open innovation* and *corporate philanthropy* frameworks, as they emphasize their ability to develop effective strategies inside an highly competitive and fluid environment. As both Porter and Chesbrough pointed out, the greatest attention in this context has to be paid to business models. But not only "open" and free software projects have a place in the enabling environments: Google, whose software and service is highly proprietary, is a striking example. Its place in the environment of the Net is indisputable, as the role of Microsoft in desktop operating systems. However, even in monopolies, once an open competitor shows up, continuing to play proprietary means paying a price, since open competitors enjoy an advantage. An example comes from the competing web servers Microsoft's IIS and Apache, and another from the server segment of the operating systems market. This could happen even to Google's search engine when social bookmarking and folksonomies projects will reach a critical mass.

Enabling environments, as free software, are already in many contexts the mainstream environment (Shankland 2004). In the future we will see more and more commercial and non commercial, market and non-market projects get together building an environment that enables their ability to pursue specific goals, that can be profit-driven or not, through both cooperation and competition.

Those reflections rise some issues. Among others: will the nature of the firm change while interacting with non-market competitors and partners? How will the community react to the growing presence of profit-driven actors? What will be the role of capital? On the social side, will the concept of "work" change? In the industrial era people's identity has been determined by their position and role in the rigid industrial value chain. How will the identity and position of people in the information society change according to their role in the *knowledge* value chain?

References

- Harald T. Alvestrand, *RFC 3935, A Mission Statement for the IETF*, October 2004
- Juan-José Amor-Iglesias, Jesùs M. Gonzàlez-Barahona, Gregorio Robles-Martìnez, and Israel Herràiz-Tabernero *Measuring Libre Software Using Debian 3.1 (Sarge) as A Case Study: Preliminary Results; ;* http://www.upgrade-cepis.org/issues/2005/3/up6-3Amor.pdf
- Jonathan Bennett, *Berners-Lee calls for Net neutrality* CNET News.com May 23, 2006; http://news.com.com/2100-1036_3-6075472.html
- Yochay Benkler, *Coase's Penguin, or Linux and the Nature of the Firm, Yale Law Journal*, 2002, *369* *http://www.yalelawjournal.org/pdf/112-3/BenklerFINAL.pdf* , http://www.benkler.org/CoasesPenguin.PDF
- Yochay Benkler, *The wealth of networks: How Social Production Transforms Markets and Freedom,* Yale University Press, 2006; http://www.benkler.org/wealth_of_networks/index.php/Main_Page
- Lorenzo Beussi, *Analysing the technological history of the Open Source Phenomenon: Stories from the Free Software Evolution*, Working paper 2005, http://opensource.mit.edu/papers/benussi.pdf
- Scott Bradner, ed, *RFC2418: IETF Working Group Guidelines and Procedures,* September 1998; http://rfc.net/rfc2418.html
- Steve Bratt, *Worldwide Participation in the World Wide Web Consortium, W3C,* May 2006, http://www.w3.org/2005/02/W3C-Global-Focus.html
- Fredrick P. Brooks, *The mythical Man-Month, Essays on Software Engineering,* Addison-Wesley 1975 (2nd ed. 1995)
- Gabriella E. Coleman, *Three Ethical Moments in Debian,* 2005, SSRN: http://ssrn.com/abstract=805287
- Henry Chesbrough, *Open Innovation,* Harvard Business School Press, 2003
- Michael Cusumano, *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad,* Free Press, 2004
- Mary Ann Davidson, *When security researchers become the problem*, News.com July 27, 2005 http://news.com.com/When+security+researchers+become+the+problem/2010-1071_3-5807074.html
- Giovanna Devetag, Enrico Zaninotto *The imperfect hiding : some introductory concepts and preliminary issues on modularity.* ROCK Working Papers. 2001 http://eprints.biblio.unitn.it/archive/00000388/01/DZModularity.pdf
- Pino Fondati, *Il Pinguino corre, ma la strada è ancora in salita*, Il Sole 24 Ore, 12 aprile 2006, http://www.ilsole24ore.com/fc?cmd=art&codid=20.0.1899767783&chId=30&artType=Articolo&DocRulesView=Libero
- Jesús M. González-Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, Vicente Matellán Olivera. *Counting potatoes: the size of Debian 2.2* http://opensource.mit.edu/papers/counting-potatoes.html
- Toshihiko Hayashi, *Fostering globally accessible and affordable ICTs*; International Telecommunication Union, 2003; http://www.itu.int/osg/spu/visions/papers/accesspaper.pdf
- Juho Lindman, *Effects of Open Source Software on the Business Patterns of Software Industry*, Helsinki School of Economics Department of Management, 2004, http://opensource.mit.edu/papers/lindman.pdf
- Ching-Chyi Lee, Jie Yang, *Knowledge value chain*, Journal of management development, vol 19, n.9, 2000
- Narduzzo A., Rossi A., *The Role of Modularity in Free/Open Source Software Development*, in S. Koch (ed by), 2005, http://opensource.mit.edu/papers/narduzzorossi.pdf
- Gregorio Robles, Jesus M. Gonzalez-Barahona, Martin Michlmayr *Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian;* http://www.cyrius.com/publications/robles_barahona_michlmayr-evolution_participation.pdf
- Francesco Rullani, *Dragging developers towards the core. How the Free/Libre/Open Source Software community enhances developers' contribution*, December 2006; http://freesoftware.mit.edu/papers/rullani_comsocenv_v5.pdf

- Dan MacLean, ed., *Internet Governance: A Grand Collaboration*, United Nations ICT Task Force, 2004
- David Messerschmitt, Clemens Szyperski, *Software Ecosystem – Understanding An Indispensable Technology and Industry,* MIT Press, 2003
- Eben Moglen, *Die Gedanken Sind Frei*, Berlin June 10th 2004 – Wizard of OS 3 speech http://emoglen.law.columbia.edu/publications/berlin-keynote.html
- Gina Neff, David Stark; *Permanently Beta: Responsive Organization in the Internet Era*, September 2002 http://opensource.mit.edu/papers/neff-stark.pdf
- Tim O'Reilly, *Compact Web2.0 definition*, 2005, http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html
- Anthony Senyard, Martin Michlmayr, *How to Have a Successful Free Software Project,* Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004; http://opensource.mit.edu/papers/senyardmichlmay.pdf
- Sulayman K. Sowe, Athanasis Karoulis, Ioannis Stamelos, *A Constructivist View of Knowledge Management in Open Source Virtual Communities,* in Antonio Dias de Figueiredo, Ana Paula Afonso, *Managing Learning in Virtual Settings: The Role of Context,* Idea group, 2005. http://sweng.csd.auth.gr/~sksowe/Publicat/IDBK001-C16_285-303_.pdf
- Sulayman K. Sowe, Ioannis Stamelos, Lefteris Angelis; *Identification of Knowledge Brokers that Yeld Software Engineering Knowledge in OSS projects,* Information and Software Technology, In press, 2006, avaliable online.
- Michael E. Porter, *Strategy And the Internet*; Harvard Business Review, March 2001
- Michael E. Porter, Mark R. Kramer, *The Competitive Advantage of Corporate Philanthropy*, Harvard Business Review, December 2002
- Mohanbir Sawhney, Emanuele Prandelli, *Communities of Creation, managing distributed innovation in turbulent markets,* California Management Review, vol.42, n.4, summer 2000
- Richard A. Spinello, *The case against Microsoft: An ethical perspective*, Business Ethics: A European Review, Volume 12, Issue 2, Page 116-132, Apr. 2003
- Stephen Shankland IDC: *Linux is now mainstream* ; CNET News.com December 16, 2004 http://news.zdnet.co.uk/software/linuxunix/0,39020390,39181356,00.htm
- Ilkka Tuomi, *Networks of creation*, Oxford University Press, 2002
- Jim Turley, *Embedded systems survey: Operating systems up for grabs*, Embedded Systems Design 2005, http://www.embedded.com/showArticle.jhtml?articleID=163700590
- Steven J. Vaughan-Nichols *Mr. Dell opens up about Desktop Linux,* March 7th 2006 http://www.desktoplinux.com/news/NS3822185143.html
- Von Krogh, G., Ichijo, K., & Nonaka, I. *Enabling knowledge creation: how to unlock the mystery of tacit knowledge and release the power of innovation.*, Oxford University Press, USA, 2000
- Greg Wallace, *Open Source: Changing the Enterprise Software Supply Chain for Good*, Linux.sys-con, Feb 2006, http://linux.sys-con.com/read/173425.htm