# Capability Coordination in Modular Organization: Voluntary FS/OSS Production and the Case of Debian GNU/Linux

**Giampaolo Garzarelli***
Università degli Studi di Roma, "La Sapienza"
Dipartimento di Teoria Economica
e Metodi Quantitativi per le Scelte Politiche
Piazzale Aldo Moro, 5
00185, Rome, Italy
ggarzarelli@tiscali.it

**Roberto Galoppini**
ACME Solutions, s.r.l.
Via Carmelo Maestrini, 98
00128, Rome, Italy
galoppini@acmesolutions.it

*Corresponding author.

**•SECOND DRAFT•VERSION 2.8.1•**
November 19, 2003
*Comments Welcome*

A very preliminary version of this paper (differently titled) was presented at the **2002 DRUID Summer Conference**, Copenhagen, Denmark, June 6-8; the gist of the idea was also illustrated at the October 14, 2002 workshop "**Advancing the Research Agenda on Free/Open Source Software**," European Commission, Brussels, Belgium. We thank participants at both meetings, in particular Maria Brouwer, Benjamin Coriat, and Aija Leiponen. Special thanks to Bdale Garbee, Brian T. Kench, Richard N. Langlois, Alessandro Rubini, Richard M. Stallman, and David A. Wheeler. Views, naïvetés, and errors are only of the authors.

Sections 1, 4, 5, and 6 were written by Giampaolo Garzarelli; sections 2 and 3 were jointly written.

# Capability Coordination in Modular Organization: Voluntary FS/OSS Production and the Case of Debian GNU/Linux


## Abstract

The paper analyzes voluntary Free Software/Open Source Software (FS/OSS) organization of work. The empirical setting considered is the Debian GNU/Linux operating system. The paper finds that the production process is hierarchical notwithstanding the modular (nearly decomposable) architecture of software and of voluntary FS/OSS organization. But voluntary FS/OSS project organization is not hierarchical for the same reasons suggested by the most familiar theories of economic organization: hierarchy is justified for coordination of continuous change, rather than for the direction of static production. Hierarchy is ultimately the overhead attached to the benefits engendered by modular organization.



## JEL Codes

D23, L22, L23, L86, M11.

## Key Words

Modularity, hierarchy, capabilities, coordination costs, software.

# 1 / Introduction

Many are accustomed to the idea that software is a product just like any other: it is purchased and it cannot be modified according to need. In short, software is perceived to be a closed system like a home appliance, such as a toaster, microwave oven or dishwasher. The challenge to this traditional idea is Free Software/Open Source Software (FS/OSS). Unlike proprietary software, FS/OSS is (often) distributed free of charge and, most importantly, entails the free/open access to the source code for purposes of study or modification. The fact that FS/OSS can be modified or studied by anyone who has the interest and the ability to do so has spawned a large community of programmers sharing not only FS/OSS, but also the capabilities to do so.[1]

We discuss FS/OSS from the organizational standpoint. Previous organizational contributions (**Benkler 2002; Garzarelli 2004**) built on a Coase Theorem-like argument (Coase 1960). They emphasized that for FS/OSS altruistic motivation is neither a necessary nor a sufficient condition for its voluntary organization. Their basic conclusion was that the raison d'être and stability of FS/OSS organization of work can be ascribed to the sharing of a common capability base among software programmers.

The present essay intends to extend these previous efforts by focusing on the coordination of work under a specific FS/OSS production regime: the

[1] Capabilities refer to "the knowledge, experience, and skills" of an organization (Richardson 1972, p. 888). See also, inter alia, Chandler (1992), Langlois (1992, 1995, 1998, 2002) and Langlois and Robertson (1995).

voluntary one. In many cases FS/OSS programmers are in fact volunteers: their work on some FS/OSS project is without remuneration, and it is not usually imposed from above, as in a typical top-down managerial hierarchy. The empirical setting considered is the Debian GNU/Linux operating system.[2]

Contrary to our expectations, we find that the production process of Debian is hierarchical notwithstanding the modular architecture of software and of voluntary FS/OSS organization.[3] In our story, however, the hierarchy rationale of FS/OSS organization is not coextensive with that of more familiar theories of economic organization: hierarchy is justified for coordination of continuously generated change, rather than for the direction of a steady (predictable) flow of production. The supplementary benefits that a voluntary FS/OSS production regime yields are, all things considered, not a free lunch, or, rather, a *free beer*[4]: hierarchy, we shall argue, is the overhead of modular organization.

But perhaps the most succinct way to convey our argument is in terms of externalities (cf. Langlois 2002). In voluntary FS/OSS production, hierarchy primarily exists to internalize the externalities that the organization of

---

[2] GNU is a recursive acronym that stands for "GNU's Not UNIX"; it is pronounced "guh-NEW" (‹**http://www.gnu.org/**›).

[3] Note that in this work we consider modularity to be tantamount to *near decomposability* (Simon 1998[1962]). That is, we have in mind a system composed of different modules the interactions among which – although tending to zero in the limit – have a nonnegligible effect on the system as a whole. This implies that not all modular systems are perfectly decomposable. In turn, to Simon this meant that near decomposability practically implies hierarchy *ex hypothesi*. We shall reach the same destination by a somewhat different trail. See also Alexander (1964) and the more recent Brusoni and Prencipe (2001) and Langlois (2002).

[4] "'Free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer'" ‹**http://www.gnu.org/philosophy/free-sw.html**›.

production *innately* generates. More precisely, hierarchy is necessary to coordinate those *Pareto-relevant externalities* triggered by the gains from trade among self-selected programmers.[5]

## 2/ Background

When we think about software we usually think in terms of "programs," that is, applications (word processors, browsers, mail-clients, and so on) that we can run on our computer to get a variety of tasks done. And yet, it must be emphasized that in these cases we use programs only in their executable binary form (object code), i.e., a machine language that the computer is able to read and process. Programs are in fact written in different computer programming languages (BASIC, C, Java, etc.) that are readable to humans. When programs are in their human-readable form they are called source code. Other types of programs, called interpreters and compilers, are used to translate the source code into its executable form. When programs are in their executable form, all users can exploit their various functions.

We may divide programs into two types: proprietary and FS/OSS. If a program is proprietary its license agreement doesn't permit anyone to copy, distribute, or modify it. In addition, most of the times the source code is not

[5] An externality caused by a party that generates the will on behalf of another (benefited or damaged) party to directly take action through exchange, agreement, compromise, bargaining, collective action and the like in order to internalize it is known as Pareto-relevant. An externality that does not generate such will is Pareto-irrelevant (Buchanan and Stubblebine 1962, esp. pp. 373-4).

even available for mere reading or studying. There even exist cases where proprietary licenses prohibit some uses for a program. One such case is that of Bitkeeper from Bitmover, Inc. Bitkeeper is a version management tool the license of which does not permit its use to individuals who are involved in developing or selling products with similar features.[6]

On the other hand, FS/OSS programs are covered by free software or open source licensing schemes. These licensing schemes specify that the source code is available, can be freely copied, modified, and distributed. Accordingly, the word "free" does not always mean gratis: the word is derived from freedom – hence, free/open access to program code. Many companies in fact sell FS/OSS for profit worldwide. For example, MySQL AB is a company founded in Sweden (now having several offices worldwide) that sells the FS/OSS MySQL database.

FS/OSS licensing schemes may (at this stage) be distinguished into two broad categories: Copyleft and non-Copyleft. A Copyleft license works this way: if modified versions of or works derived from programs protected under Copyleft are distributed, they must be relicensed under the same license. This is sometimes referred to as persistence of license. For example, GNU General

---

[6] Section 3 (Licensee Obligations), paragraph (d), of the *BitKeeper License version Sep-16-2002* states: "Notwithstanding any other terms in this License, this License is not available to You if You and/or your employer develop, produce, sell, and/or resell a product which contains substantially similar capabilities of the BitKeeper Software, or, in the reasonable opinion of BitMover, competes with the BitKeeper Software" ‹**http://www.bitkeeper.com/Sales.Licensing.Free.html**›. The quote from this link is from October 28, 2003; on November 8, 2003 the link was still there, but not the *BitKeeper License*. In order to view the very last version of the license agreement the *Bitkeeper* site suggests to run a command from the installation; an older version of the license (*BitKeeper License version 1.40, 08/23/02*) can be found at ‹**http://www.bitkeeper.com/bkl.txt**›.

Public License (GNU GPL) (‹**http://www.gnu.org/copyleft/gpl.html**›), the most famous and used FS/OSS license, is copyleft. Non-Copyleft licenses might be modified and distributed as well, but it's possible to add additional restrictions to their use (e.g., software under an Apache license ‹**http://www.apache.org/LICENSE.txt**› that can be rendered proprietary).

For analytical purposes we are placing Free Software and Open Source Software under a common rubric. And the two software movements do to a large extent factually share the same rights. But there are some crucial differences.

One dimension of difference is motive (compare **Wheeler n.d.1**). The Free Software Foundation (FSF) (‹**http://www.gnu.org**›) emphasizes the possibility of using and sharing software independently from the control of others, where the objective is not technological but social (‹**http://www.gnu.org/philosophy/free-sw.html**›). To quote a FSF brochure (n.d., p. 1), "[m]ost software projects aim to make better technology. The GNU project [‹**http://www.gnu.org**›] has a higher goal: to make a better society."

The Open Source Initiative (OSI) (‹**http://www.opensource.org**›) conversely emphasizes a technological objective. It is claimed that if you adopt a license scheme that falls within the Open Source Definition **(**‹**http://www.opensource.org/docs/definition.php**›) the output will be software that is allegedly of higher "quality." This "efficiency" motivation is used to stress the commercial viability of OSS (‹**http://www.opensource.org/index.php**›).

And yet, it must be highlighted that there have been licenses that were OSS but not FS. (So far, there have been no examples of the opposite.) For instance, the original Artistic License (‹**http://www.gnu.org/licenses/license-list.html#NonFreeSoftwareLicense**›) was not considered a free software license, but an open source one (‹**http://www.opensource.org/licenses/artistic-license.php**›); Artistic License 2.0 corrected this (‹**http://dev.perl.org/perl6/rfc/346.html**›). Similarly, in its early releases, the Apple Public Software License (APSL) was approved by the OSI (‹**http://opensource.org/licenses/apsl.php**›), but rejected by FSF (‹**http://www.gnu.org/philosophy/historical-apsl.html**›). Starting with version 2.0, however, FSF endorses APSL as well (‹**http://www.opensource.apple.com/news/2.0-announce.html**›).

Note, however, that in the vast majority of cases a license that meets the OSI's open source definition also meets the FSF's free software definition. In particular, the most common FS/OSS licenses are both FS and OSS: GPL, Lesser GPL (**LGPL**), **MIT/X** and **BSD-new** (**Wheeler 2003**; **Wheeler n.d.2**).

FS/OSS is clearly a complex phenomenon. As such, it is gaining attention. This is true not only in the general domain of academic research,[7] but also in the policy and business domains.

For instance, in October 2000 the United States President's Information Technology Information Committee recommended that "the Federal

[7] See, inter alia, Peyrache et al. (2000), **Kuan (2001)**, **Benkler (2002),** Feller and Fitzgerald (2002), Lerner and Tirole (2002), **Garzarelli (2004)**, and von Hippel and von Krogh (2003).

government should aggressively encourage the development of open source software for high end computing" (‹**http://www.itrd.gov/pubs/pitac/pres-oss-11sep00.pdf**›). Furthermore, the United States Army and the Defense Information Systems Agency have commissioned the MITRE Corporation to produce a business case study (2001) and a report (2003) about FS/OSS.[8] The report, in particular, explored the feasibility of adopting FS/OSS solutions for Department of Defense (DoD) organizations, but found that much FS/OSS already plays an important role in the general functioning of the DoD. Therefore, the report continues, attempting to remove FS/OSS from current DoD infrastructure would be counterproductive; and, all things considered, it would be more productive to increase its use in some cases.

Other governments are also giving FS/OSS serious consideration. The Center of Open Source and Government for example supports the so-called South African Proposed Strategy for Using Open Source Software in the South African Government (‹**http://www.oss.gov.za/docs/OSS_Strategy_v3.pdf**›). "While government procurement policy should be neutral to ensure that governments do not introduce market distortions into the world economy," writes the Center, "there should be an appreciation of the social benefits of fostering Open Source software development in a proper Open Source Government Policy plan." (‹**http://www.egovos.org/SouthAfricanStrategy.html**›). Similarly, the daily

8 ‹**http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/index.html**› and ‹**http://www.egovos.org/pdf/dodfoss.pdf**›.

press often informs about how Germany, Japan, South Korea and China have shifted (or are interested in shifting) their public administration toward FS/OSS.

The European Union (EU) is also studying FS/OSS (**FLOSS Final Report 2002**). It is organizing (and has organized) several types of conferences and workshops to explore the FS/OSS phenomenon, and its viability for EU administrative governance, not only for the bureaucratic sphere but also for the "more democratic sphere," for narrowing the gap between citizens and various levels of government (so-called e-government). (Visit for example ‹**http://www.prelude-portal.org/3roses/index.php**›, and various links therein.)

Some hardware producers, moreover, are still rather optimistic about FS/OSS solutions for personal computers notwithstanding early disappointments. The *Times of India* has published an article on June 2, 2003 revealing that IBM will launch India's first Linux certified PC, the NetVista A30 model, at the cost of about 850 US dollars.[9]

What about Microsoft Corporation? After evaluating the potential competition from FS/OSS (‹**http://www.opensource.org/halloween**›), it decided to introduce "Shared Source" in 2001. Shared Source is a class of licenses that partially replicates some of the rights of FS/OSS. It allows those who are entitled to participate in the Shared Source program to inspect the

[9] ‹**http://timesofindia.indiatimes.com/cms.dll/xml/uncomp/articleshow?msid=2218**›.

source code and to perform debugging operations.[10] Yet, the user cannot modify, redistribute or commercialize Shared Source – the intellectual property rights remain Microsoft's. In brief, Shared Source keeps all traditional proprietary characteristics.[11]

At the same time, however, Microsoft is a consumer of FS/OSS. Windows and Office, which are key Microsoft products, include non-copylefted FS/OSS components. For example, Windows incorporates many FS/OSS components that implement Internet components (primarily from the BSD Unix-like operating systems); and, as the 2003 MITRE report also notes (p. 22), Office includes the FS/OSS *zlib* (‹**http://www.gzip.org/zlib/**›) compression software. In addition, Microsoft sells FS/OSS bundled to its proprietary one: "Windows Services for Unix" contains software licensed under GPL (such as the GNU compiler *gcc*). Clearly, Microsoft recognizes some "value" in FS/OSS.[12] However, at this time Microsoft does not sell entirely FS/OSS products to consumers.

The statistics certainly justify all this attention to FS/OSS. For example, just in the last week of October 2003 more than 3 million FS/OSS packages have been downloaded from SourceForge alone (‹**http://www.sourceforge.net/**›). At the same time, Netcraft reports that if one

---

[10] See ‹**http://www.microsoft.com/resources/sharedsource/licensing/default.mspx**› and ‹**http://www.microsoft.com/resources/sharedsource/Initiative/Initiative.mspx**›, respectively.

[11] See for example ‹**http://www.antifork.org/opensource/advocacy/shared_source.php**›.

[12] Something in sharp contrast to Microsoft's attack on GNU GPL and LGPL licenses with its "Royalty-Free CIFS Technical Reference License Agreement." See ‹**http://www.gnu.org/press/2002-04-11-ms-patent.html**›.

considers the trend in top developers, we see that from October 2003 to November 2003 Microsoft, SunONE and Zeus experienced a total loss of 2.58 percent (that translates into a net loss of 11,717,993 sites) to Apache (‹**http://news.netcraft.com/**›).[13]

Before proceeding, allow us moreover to note that FS/OSS is not at all a recent phenomenon. The origins of FS/OSS go back to the beginning of electronic computing, and its diffusion was promulgated by the so-called hacker culture. Hackers are very creative software developers who believe in the unconditional sharing of software code and in mutual help (Raymond 2001, pp. 1-17; 169-91). Apparently, the sharing ethos began mostly because the early computer users were also programmers. Computing writ large was in fact a fairly restricted activity, and many of the first computers included the source code. As a result, the few computer users shared information and programs. This changed in the 1950s and 1960s when IBM introduced the 360 system – a system that was proprietary not only in hardware but also in software.[14] As a consequence, software too became a source of profit. But since around the mid-seventies (thanks to the introduction of the MITS/Altair computer, the first true minicomputer) there has been a growing interest in the original sharing philosophy in parallel with the unprecedented diffusion of proprietary software (cf. Langlois 1992).

---

[13] For more figures on FS/OSS, see **Wheeler (2003)**.

[14] Which eventually led to *United States* v. *International Business Machines Corporation* 1956, Trade Case No. 68, 245 (S.D.N.Y. 1956).

Yet, it is thanks to the capillary diffusion of the Internet that in recent years we have growth in the number of interactions among FS/OSS programmers as well as in the popularity of FS/OSS. Further, many programs on which the Internet runs are FS/OSS: for example BIND (translating a domain name into a corresponding IP address), SendMail (the most used mail system), and Apache (the most popular web server).

## 3/ Voluntary Modular Organization and Debian GNU/Linux

The production model of FS/OSS can be idealtypically classified into three basic organizational "categories."

- Corporate.
- Hybrid.
- Voluntary.

The FS/OSS projects falling under the corporate category do not seem to have organizational attributes different from traditional proprietary projects. Indeed, the organization of work followed seems to be the same as that of a traditional firm. There is a well-defined hierarchy for the production process: there are a project manager, analysts, programmers, and end-users that within an organization are employed to – respectively – direct production and define, produce, and use the system. All stages of production are carried out by internal organization.

In a recent study, **Henkel (2003)** focuses on the corporate FS/OSS production process. One of his concrete illustrations is "Openadaptor" software. Designed and developed by the investment bank Dresdner Kleinwort Wasserstein in London, Openadaptor is a middleware platform to help the cooperation of trading and e-business systems.

The difference between the voluntary and the hybrid idealtypes is not often considered. In the voluntary case the process of production is open for all contributors: deadlines and tasks may be assigned by the organization, but programmers are not completely bound to follow them.

In the hybrid case, volunteers from outside the organization as well as remunerated programmers contribute to the production process. The tasks and timeline are set by the organization and are binding for the remunerated programmers. When the volunteers do not respect deadlines or assignment of task, the remunerated programmers will. And when there are urgent necessities for, e.g., a patch, the hybrid organization may work in house even if this means duplication of volunteer efforts from outside. The American company Red Hat Enterprise is a good example of a hybrid. Red Hat produces a distribution of the GNU/Linux operating system, which it not only freely distributes on the web but also sells, as off-the-shelf packages, with manuals and limited warranty as add-ons.

The Debian Project (‹**http://www.debian.org**›) is an association of volunteers who cooperate to create a free/open operating system, named

Debian GNU/Linux. Ian A. Murdock started the Debian project in August 1993, while still an undergraduate at Purdue University.

The raison d'être of the Project is not one of profit.[15] Even if Debian has expenses, such as registering the debian.org domain or reimbursing travel expenses for conferences, only donations are welcome; and donated funds are not used, e.g., to pay for development of software or for production of documentation. If anything, the objective function that Debian organization tries to maximize is the one that later became typical of the OSS slogan, namely, product quality.[16]

To get involved in the project volunteers are required to "abide" to the Debian Social Contract (‹**http://www.debian.org/social_contract**›) as well as to the Debian Free Software Guidelines (aka, DFSG) (‹**http://www.debian.org/social_contract#guidelines**›).[17] More specifically, to join the Debian project, an applicant (a potential contributor), needs to follow "The Debian New Maintainer Process": "a series of required proceedings to become a Debian developer" (‹**http://www.debian.org/devel/join/newmaint**›).

---

[15] "Debian motivated the formation of **Software In The Public Interest, Inc.,** a New York-based non-profit organization. SPI is a non-profit organization which was founded to help Debian and other similar organizations develop and distribute open hardware and software. Among other things, SPI provides a mechanism by which The Debian Project may accept contributions that are tax [deductible] in the United States" (‹**http://www.debian.org/doc/manuals/project-history/ch-intro.en.html**›).

[16] The "focus [is] on providing a first-class product and not on profits or returns, and the margin from the products and services provided may be used to improve the software itself for all users whether they paid to obtain it or not" (‹**http://info.astrian.net/doc/debian-history/html/apA.html**›).

[17] From which derives the previously mentioned Open Source Definition. The Debian Free Software Guidelines, which are also part of the Social Contract, define the scope of the software licenses that can be accepted for Debian packages. Indeed, there is no specific license for Debian other than the licenses in the various software packages that make up the distribution (‹**http://www.debian.org/social_contract#guidelines**›).

Through this process the applicant is identified, is required to understand and share the philosophy of Debian, and, along with the Application Manager (a mentor), the "applicant must [also] provide assurance that [he or she] can, in fact, do the job for which [he or she has] volunteered" (‹**http://www.debian.org/devel/join/nm-step4**›).

"The document of utmost importance to the organization" (‹**http://www.debian.org/devel/**›) of Debian is, however, the Debian Constitution (‹**http://www.debian.org/devel/constitution.en.html**›). The Constitution establishes a hierarchy. That is, within the Debian Project there are de jure different roles, e.g., the Project Leader, the Technical Committee, and the Developers.

In the Debian hierarchy, the role of ultimate coordinator lies with the Project Leader. For example, the Leader helps define the project's vision, lends authority to Developers and makes any decision that requires urgent action. The Leader also represents Debian outside the Project (e.g., goes to conferences and gives talks). All Debian Developers can vote to elect the Leader, and a Leader's mandate lasts one year.[18]

The charge of the Technical Committee – composed of a minimum of 4 and a maximum 8 members – is akin to that of a judging body. The Technical Committee has authority over solving divergences about such issues as the

[18] Besides the Constitution, on the Leader see also ‹**http://www.debian.org/devel/leader**›. What is also interesting in this FS/OSS organization of work is that although Individual Developers are at the bottom of the hierarchy, they can, as a Body, override a decision taken by a superior (such as the Project Leader or Technical Committee) by way of a General Resolution (where a Resolution is a voting procedure to rediscuss technical and nontechnical issues) or of an election.

contents of a reference manual or ownership of a command name. Yet, it may also make formal statements about its opinion on any matter.

The Constitution doesn't impose any obligation on anyone to work continuously on the Project; in fact, a contributor can leave the project at any time or resign from his or her position or duty by a simple announcement. There is no termination of a formal employment contract as in a firm.[19] It may be noted in this context that whereas in a firm one usually resigns for the possibility of better remuneration elsewhere, in this case one usually resigns for reasons of loss of interest or of lack of time.[20]

More interesting, perhaps, is the related issue of assignment of duty once an applicant has been accepted into Debian. The duty is in fact decided in conjunction with the Application Manager (i.e., the mentor) based on the capability of the applicant: the process explicitly acknowledges the dispersion of capabilities, favoring self-selection. "The Application Manager will work out with the applicant just which tasks the applicant wishes to volunteer to

---

[19] Recall the Simonian formal theory of the employment relationship. The employer pays a wage to the employee that grants her the right to choose which action $a \in A$ the employee will be instructed to perform, where $A$ is the set of actions the employee consents to – the so-called zone of indifference or zone of acceptance (Simon 1951). Accordingly, in a firm the employee is always in a situation of subordination (for reasons of remuneration) in that, unlike the market, the ultimate decision rights are alien to him. In voluntary FS/OSS organization such as Debian, authority figures like a Project Leader are granted privileges from reputation, rather than rights by formal contracts (e.g., **Garzarelli 2004**). In some cases, some reputation privileges may stay even after such figures exit a project. One case of this is that of Richard M. Stallman and GCC (GNU C Collection – was GNU C Compiler). GCC was originally written by Stallman; currently, Stallman is not even listed in the GCC Steering Committee (founded in 1998), but he is still recognized as the father of the Project. Visit ‹**http://gcc.gnu.org/**›.

[20] Strictly speaking, the flip side, i.e., the de jure invitation to leave (being dismissed) is contemplated, but it is de facto never implemented on the basis of technical reasons. It could be implemented for what we may call behavioral reasons (called Debian Machine Usage Policies): for example, using the email address of the association for personal and not for job-related reasons. This is clearly stated in ‹**http://www.debian.org/devel/dmup**›.

perform" (from step 4 of the Debian New Maintainer process ‹**http://www.debian.org/devel/join/nm-step4**›).

The history of Debian, which also describes the purposes of the project, sheds some light on matters organizational as well (‹**http://info.astrian.net/doc/debian-history/html/index.html#abstract**›) – especially its appendix A, the so-called Debian Linux Manifesto, written on 6 January, 1994 by Debian founder Murdock (last revised 29 December, 1999). During 1994, when Debian was already to the 0.91 version of the system, it was decided to devote some efforts to organize the Debian Project, since there were already people involved and organization was becoming an issue (Murdock was still putting together the releases by himself at that stage (!), see ‹**http://info.astrian.net/doc/debian-history/html/ch4.html**›). The following excerpt makes clear the motivation behind Debian's free/open organization.

> By involving others with a wide range of abilities and backgrounds, Debian is able to be developed in a modular fashion. Its components are of high quality because those with expertise in a certain area are given the opportunity to construct or maintain the individual components of Debian involving that area. Involving others also ensures that valuable suggestions for improvement can be incorporated into the distribution during its development; thus, a distribution is created based on the needs and wants of the users rather than the needs and wants of the constructor. It is very difficult for one individual or small group to anticipate these needs and wants in advance without direct input from others (‹**http://info.astrian.net/doc/debian-history/html/apA.html**›).

So, the free/open structure of Debian organization draws its rationale from the (typically Hayekian[21]) belief that specific productive knowledge is not held in its entirety by a single mind. But rather than seeing the capability dispersion problem as a constraint, as is often the case (or, rather, the rule), the Debian philosophy of production sees it as an opportunity: it exploits dispersed capability to its advantage by letting productive knowledge search its organization, rather than vice versa.

But let us be clear at this juncture about what "free/open" means to Debian. The self-selection mechanism is composed of the following ingredients: spare time, some technical knowledge, and an interest in the project; but the New Maintainer process (‹**http://www.debian.org/devel/join/newmaint**›) and the Constitution prevent the participation of unskilled or bothersome participants. Debian hence holds the ultimate right to select which capabilities should join its mission.

As is usually the case, however, there is more to organization than what is spelled out in its statute or chronicled in its official history. Next to the formal Debian hierarchy there exists the informal one. As is the case in professions and academia,[22] reputation plays a fundamental role in the FS/OSS world, and Debian is no exception. In actual fact, the positive and negative externalities deriving from reputation often have precedence over formal

[21] Hayek (1945, Chs. 2 and 4) and cf. Raymond (2001).

[22] In fact, organizations sharing similar characteristics are the professions (Savage and Robertson 1999; **Garzarelli 2004**) and in open science (David, e.g., 1998).

hierarchy. It is possible for someone with no formal authority to have authority over others because of positive reputation; and it is possible for someone having formal authority over others to lose it because of negative reputation. Bdale Garbee, a former Debian Project Leader and continuing member of the Technical Committee, put it this way in a recent email (25 July, 2003) to one of the authors.

> I believe the hierarchy of 'authorities' who have the ability to hasten the selection or rejection of changes that may have broad impact is only vaguely related to the visible hierarchy of people with titles in the project. While in theory the Technical Committee, for example, holds a lot of responsibility, in practice almost every major change that I can remember was shepherded along by one or two key contributors in that area of the Project who were generally accepted as 'masters' of that problem domain in the project, often without having any particular title in the organization.
>
> So, in some sense, I believe that it may be accurate to describe a hierarchy of technical authority in a project like Debian, but it may not be a visible hierarchy, or the specific visible hierarchy that someone on the outside of the project would 'naturally' assume.

In practice, this means that leadership (which translates directly into formal and informal hierarchy and authority) is not just established in bureaucratic or rational fashion, but in charismatic fashion as well (Weber 1964[1947], pp. 358ff.); here, charismatic authority mostly derives from earned respect often proven by leading a big, successful project. As a matter of fact, charismatic authority may be, in some circumstances, more "efficient" than authority deriving its legitimacy from well-established rules (Coleman 1990; Langlois 1998). We return to this point below.

Therefore, five organizational characteristics of the voluntary Debian Project stand out.

- There is no profit motive driving the organization.
- There is no contractual employment relationship.
- The "out" mechanism of contributors from the Project is self-selected as in a traditional market reality: factually, there is complete self-selection.
- The "in" mechanism of contributors to the Project is as in a traditional corporate reality. In other words, a developer who wishes to contribute cannot do so automatically, but will be selected by Debian: there is only partial self-selection.
- Debian presents elements of formal and informal hierarchy/authority.

It is clear that these characteristics render voluntary FS/OSS organization uncommon. From the perspective of the present argument, however, the most interesting organizational characteristic is the last.

## 4/ The Problem of Economic Organization

The problem of economic organization emerges when there is a need to coordinate different human capabilities and comparative advantages to accomplish a variety of tasks in order to achieve an end. As such, the first problem is the discovery of different comparative advantages and human

capabilities so as to organize them in a productive fashion (Knight 1967[1933], pp. 6 and 17).

Discovering knowledge differences or similarities and making use of them in order to pursue one's ends is however not costless. Knowledge search is expensive. Because knowledge is dispersed, the organizational problem does not usually have a cheap and easy solution. Ideally – think of the market – the "first-best solution" to the problem would result in the coincidence (or collocation) of productive knowledge and rights to act on that knowledge in the same hands. As in the case of Mohamed and the mountain whereby Mohamed could go to the mountain or the mountain to Mohamed, there are two ways we could achieve a solution to this problem. We could let capability move to those who have the rights to act; or, we could let rights to act move to those with capability. The perfect migration, as it were, of knowledge to rights and rights to knowledge is possible only in markets, for in firms perfect alienation of rights does not exist. A worker in a firm cannot sell (or exchange) his or her job (the employment bundle of rights) or the machinery he or she works with to someone else and capture proceeds from such alienation; similarly, the worker's capability does not always match his or her assigned task. Free entry and exit in a firm is in fact limited by its rules, which are not based on voluntary exchange of rights (Jensen and Meckling 1992; **Garzarelli**

2004)[23]. And hierarchy (no matter how thick or well-defined) is one manifestation of this.

Take for instance the agency problem, one of the most influential justifications for the existence of the firm (e.g., Alchian and Demsetz 1972). The problem to solve in this account is that of finding the efficient management of agents through time. Since the firm entails team production in which final output is not divisible in terms of individual input, it is impossible to control shirking. We thus have externalities that can only be governed by delegating production control to a principal. But who supervises the principal? By granting the principal the status of residual claimant, one solves the problem of the principal's incentives. We obtain in this fashion the classical capitalist firm presenting an acceptable amount of shirking.

In voluntary FS/OSS production, we saw, agency is not much of a problem because people are self-motivated. When there are a large number of potential agents, one can in fact solve the principal-agent problem by self-selected sorting as in Debian. That is to say that one lets agents be aligned with the problems they are most inclined to solve anyway – in a way, one strives to let people be their own principals. And if many constituents of the same organization are at once principals and agents, then they to a large extent also

[23] See Gifford (1991) and Vanberg (1992) for how firm and market can be distinguished by their different rules. Chester Barnard and Philip Selznick may have been the first to focus on the constitutional element of organizations (see Langlois 1998). Alchian's (1977[1965], pp. 137-9) distinction between private and public ownership rights is also germane.

share the same rights, and this leaves, all other things constant, little room for a shirking justification of hierarchy.[24]

Consider next the market versus hierarchy approach to the boundaries of the firm (Williamson, e.g., 1991), arguably the most sophisticated attempt to formalize (or "operationalize," as Williamson put it in several contributions) the ideas presented by Coase (1937). Putting matters at their simplest, in this approach a transaction that is characterized by high asset specificity is considered to be in the domain of hierarchy, whereas a transaction with low asset specificity is considered to be in the domain of market. Asset specificity fundamentally refers to the lack of fungibility of some tangible and intangible productive assets.

Asset specificity opens the door to opportunistic behavior, namely, the attempt to maximize profit (and possibly utility) through guile. When because of asset specificity there arises the possibility of opportunistic behavior by at least one party engaged in a transaction, there also exists a probability of lock-in to a sterile bilateral monopoly situation that may lead to a tussle for Marshallian quasi-rents (cf. Alchian and Woodward 1988, pp. 67ff.). When this is the case – and this is the crucial point – traditional market contracts are not sufficient to protect against opportunistic behavior, and this can lead to vertical

[24] For instance, Eric Raymond, hacker and author of the very influential open source "manifesto" *The Cathedral and the Bazaar* (2001), writes: "the poor beleaguered conventional manager is not going to get any [succour] from the *monitoring* issue; the strongest argument the open source community has is that decentralized peer review trumps all the conventional methods for trying to ensure that details don't get slipped" (p. 59, original emphasis).

integration (also called by Williamson hierarchy or intentional governance structure) (cf. Klein et al. 1978).

The bottom line of this approach is to create asymmetric rights among parties transacting among technologically separable interfaces: transaction cost theory substitutes the familiar technological and production marginal cost calculation with a marginal cost calculation of devising, executing, and supervising task completion. As a result, the distinction between hierarchical and nonhierarchical organization is here solved by asset specificity considerations: the nonhierarchical structure (market) is the residual of the transaction-cost calculation done within the hierarchical structure (firm).

Voluntary FS/OSS organization does not seem to present transactions with high specificity: software is supple and easy to reorganize compared to, say, automobile construction where the process of production involves machinery with high specificity, such as dies. Computers too are highly fungible.[25] Or, more precisely, the specific assets in FS/OSS production are mostly knowledge (different capabilities or skill levels) and spare time.[26] But, in the main, these specifities are both self-selective: knowledge "enters" its most valued use spontaneously when it desires, and it "exits" in exactly the same

[25] In fact, software and computers are General Purpose Technologies (GPTs) (Helpman and Trajtenberg 1998). This does not at the same time mean that the relationship between low transaction cost and GPT is necessarily one-to-one – but this impression requires further investigation.

[26] Take note that knowledge and spare time specificities are not generally incompatible with Williamson's (1991, p. 281) "human and temporal specificities." But, as pointed out before long, Williamson's categories of specificity can be considered as a subset of the ones considered here since they deal with static scenarios (with transaction cost as the friction of economic systems in the Arrovian sense) – hence they are not always sufficient conditions for hierarchy. Cf. Langlois and Robertson (1995) and the Fisher body symposium in the April 2000 issue of the *Journal of Law and Economics* 43(1).

fashion, "marginalizing" lock-in scenarios. This is possible because programmers share a common knowledge base. The knowledge sharing assures that the "checks and balances" in the process of production are implemented by knowledge externalities, a way to communicate behavior and reputation of programmers. It would consequently be irrational to act opportunistically if one hopes to stay in the trade and possibly advance in "rank." The same logic applies if a volunteer is in the trade for so-called career concerns, i.e., in order to get a better paying job later on (cf. Lerner and Tirole 2002).

Let us now also briefly consider a third approach to economic organization that derives its positive heuristic from both previous accounts: the "property right" (Hart, e.g., 1996). In this incentive-based theory, the firm is a collection of assets the incomplete contract structure of which generates two types of rights: specific and residual. As their names imply, specific rights are those that can be more or less defined; while the residual ones are not predefined. Here, the problem to solve is the wealth-maximizing allocation of residual rights. As in Williamson, when there's fungibility of assets problems of opportunism are nonexistent. If the opposite situation prevails, then the most "efficient" course of action is to assign ownership of (tangible nonhuman) assets to one party. The party who will be the owner will be the one who will maximize the joint rents deriving from integration. The issue of the firm boundaries, in other words, is one of ownership: the allocation of property rights falls into the hands of those who are the most efficient investors.

Consequently, when there's a reallocation of property rights, there's also a change in incentives to invest. We thus have a precise theory of the boundaries between hierarchical and nonhierarchical organization that incorporates the insights of agency and transaction cost theories: authority is necessary because – given incomplete contracts – there's a necessity to efficiently allocate residual rights of control in order to generate and maintain incentives for cooperation among parties.

Analogously to its inspiring agency and transaction-cost approaches, the property right theory does not seem to capture the essence of voluntary FS/OSS production. Basically, in FS/OSS there are no assets to own – or rather, no one is allowed to appropriate anything. Or, to make the same point through a different framework, we don't have a commons or an anticommons tragedy in voluntary FS/OSS production that needs to be governed for there is no ownership problem.[27]

A voluntary FS/OSS regime such as the Debian slips through the meshes of some traditional organizational theories like a ghostly anomaly. As we saw, voluntary FS/OSS production stands the organizational problem on its head: it solves the problem of organization by freely allowing rights to go to capability as well as capability to go to rights. This freedom reduces search

[27] A tragedy of the commons arises when inefficient specification of property rights leads to an overutilization of resources (think of fisheries and oil exploitation), and in the extreme to their exhaustion. The tragedy of the anticommons is instead the symmetric case. That is, property rights are so meticulously specified as to practically create no productive use of resources: anyone can block someone else's resource exploitation (to use an imagery of legal doctrine, imagine that everyone possesses the rights to an essential facility but all deny access). Generally see the recent Buchanan and Yoon (2000), and cf., more specifically, **Benkler (2002)**, which introduces the model of "commons-based peer production."

costs, and to a large extent renders the organization of the source code and that of the programmers the same problem.[28]

We would seem to have, in different terms, a "best of all possible worlds" scenario – as is often more or less explicitly assumed in neoclassical production theory[29] – where process of production and outcome of production coincide. If knowledge about output is isomorphic to the organization of its input, then we would expect hierarchy to disappear. But if this may be true in a static context – as the theories of our pageant above would ultimately imply – it need not necessarily be true in a (seldom-considered) dynamic context.

Simply put, a dynamic context is one foremost characterized by innovation and mistake-ridden learning. One often forgets that it is the combinatorics of heterogeneous productive knowledge (or capabilities) that leads to such characteristics. In voluntary FS/OSS this is *a fortiori* so given also that in a free/open organizational architecture heterogeneity of knowledge to a large extent coincides with equal rights of decision and alienation. This sets off greater experimentation and variety – hence greater innovation potential. And as articulated perhaps in the most parsimonious fashion by Radner (1992), in such uncertain environment rests a solid rationale for hierarchy (and authority) as well.

Viewed differently, organization – or, if you like, instances of nonmodularity, namely, the partitioning of alienation rights from decision

[28] Cf. Torvalds (1999, p. 108).

[29] See Winter's (1982) already-classic statement.

rights – also comes about for managing various types of interactions among capabilities and innovation. Indeed, organization and technology often coevolve: the substantive content of the capabilities that compose organization pushes innovation; and innovation pushes organization to diversify its capability base.[30] When modularity is present organizational boundaries not only change thanks to their internal coevolution, but also thanks to their external coevolution generated by their high degree of porosity. And, to maintain organizational coherence or stability, coevolution needs its governance structure, too. (More on this in the next section.)

The problem of economic organization then has two facets: static and dynamic. And although with its emphasis on incentive conflicts, hazardous and inefficient transactions, and practically perfect production knowledge the static facet has attracted considerably more attention, both facets have but only one solution: hierarchy.[31] Indeed, with learning effects and growth in the extent of the market (i.e., as capabilities diffuse), one could quite convincingly make the case that hierarchical solutions in static contexts are nothing more than a subset or a path-dependent manifestation of hierarchical solutions in dynamic ones (Langlois, e.g., 1995).

One implication of this is to see transaction cost in less static terms, that is, as a cost that also foremost arises from heterogeneous knowledge (Dahlman, e.g., 1980, pp. 79ff.). We may think of this type of transaction cost as the cost of,

[30] On coevolution see for example the recent Nelson and Sampat (2001).

[31] However, compare Williamson's work until 1975, and the more recent Williamson (1999).

e.g., bargaining with, coordinating with, organizing, informing, teaching, and persuading others. In light of this, capabilities and transactions – production and exchange – become indissoluble, if orthogonal (Langlois and Robertson 1995).

## 5 / Software Programming as Dynamic Organization

The reason for the persistence of some (formal or informal) hierarchy (or authority/guidance/principal) in such projects as Debian therefore lies in the costs tied to the advantages of free/open organization. And these costs (that are more than outweighed by benefits) must concern, we believe, transactions coupled to capability reorganization in a dynamic production context. It is now incumbent on us to identify the locus of these costs.

Software programming is generally not an easy task. It presents a high degree of uncertainty. For example, not only does it involve the mere creation of code, but often also the aspect of making the code compatible with other codes. This is particularly true in more recent times as computers (even those using different operating systems) have become increasingly networked. The necessity to render programs compatible entails that software undergo more changes than traditional knowledge-intensive products. Thus new problems are always emerging, necessitating wide scope for testing of multiple solutions.

This naturally leads to a substantial amount of experimentation and to a need to frequently coordinate trial-and-error learning.[32]

To overcome this "complexity constraint" (Baetjer 1998, p. 32) – i.e., to allow for experimentation and learning among different solutions – programs, especially modern ones of the object-oriented type, are organized into different, interacting modules. It is thus possible to change a part of a module or an entire module without knowing all information about the program that the module belongs to and without altering other modules or the overall purpose of the program. In a context of frequent and rapid change such as the software one, this allows for the coordination of disparate software codes, or, if you like, for the coordination of disparate knowledge structures, and therefore for the coordination of learning. The testimony of Linus Torvalds (1999, p. 108, original emphasis), creator and principal software architect of Linux, is in this regard most eloquent.

> With the Linux kernel it became clear very quickly that we want to have a system [that] is as modular as possible. The [FS/OSS] development model really requires this, because otherwise you can't easily have people working in parallel. It's too painful when you have people working on the same part of the kernel and they clash.
>
> Without modularity I would have to check every file that changed, which would be a lot, to make sure nothing was changed that would effect anything else. With modularity, when someone sends me patches to do a new filesystem and I don't necessarily trust the patches *per se*, I can still trust the fact that if nobody's using this filesystem, it's not going to impact anything else. … The key is to keep people from stepping on each other's toes.

[32] On coordination of trial-and-error learning, see especially Nelson and Winter (1977) and Langlois (1992).

Thanks to its modular structure a program hides information among modules while at the same time allowing for their communication – this principle is known as *information hiding* (Parnas 1972).[33] Parnas et al. (1985, p. 260) say that according to this principle "system details that are likely to change independently should be the secrets of separate modules; the only assumptions that should appear in the interfaces between modules are those that are considered unlikely to change." Information hiding assures that software is extendible, compatible and reusable. In actual programming practice, it is a very efficient way to reduce the appearance of bugs. Information hiding moreover suggests that the primary benchmark to assess the efficiency of a particular software is not so much the ability to perform tasks as the ability to evolve to potentially perform tasks even better (Baetjer 1998).

Information hiding is a principle that actually derives from a more general characteristic of modularity known as *encapsulation*. "Encapsulation of information," assert Mark S. Miller and K. Eric Drexel in "**Markets and Computation: Agoric Open Systems**," a classic contribution on the (Hayekian and Coasean) economic properties of computation,

> ensures that one object cannot directly read or tamper with the contents of another; communication enables objects to exchange information by mutual consent. The encapsulation and communication of access ensures that communication rights are similarly controlled and transferable only by mutual consent. These properties correspond to elements of traditional object-oriented programming practice; in large systems, they facilitate local reasoning about *competence* issues – about what computations the

[33] But compare Brooks (1975, pp. 78ff.).

> system can perform. Extending encapsulation to include computational resources means holding each object accountable for the cost of its activity ... . In large systems, these extensions facilitate local reasoning about *performance* issues – about the time and resources consumed in performing a given computation. …
>
> [Encapsulation] establishes protected spheres in which [individuals] can plan the use of their resources free of interference from unpredictable external influences. This enables [planning and acting] despite the limited, local nature of most knowledge; it thus permits more effective use of divided knowledge, aiding the division of labor. The value of protected spheres and local knowledge has thus far been the sole motivation for giving software modules 'property rights' through encapsulation. … Encapsulation and communication of resources correspond to ownership and voluntary transfer, the basis of trade. … [M]otivated by the need for decentralized planning and division of labor, computer science has reinvented the notion of property rights (**Miller and Drexel 1988**, original emphasis).[34]

Encapsulation – which we may liken to division of labor and specialization benefits arising from the clear definition of property rights – is therefore fundamental for the software production process: the clear separation of mine from thine through the creation of common standards is the cradle of program evolution.[35]

Encapsulation favors *speciation* (Houthakker 1994[1956]) along many dimensions. Speciation occurs not just in the space of final output, but also in the spaces of interrelations among the modules and of inputs.[36] Essentially, by

---

[34] All references to this article are to the webbed version.

[35] In his classic discussion of property rights, Alchian (1977[1965], p. 140, original emphasis) highlighted the following benefits of property encapsulation: "(1) concentration of rewards and costs *more* directly on each person responsible for them, and (2) comparative advantage effects of specialized applications of (a) knowledge in control and (b) of risk bearing." Langlois (2002) has just developed a full-blown modular theory of organization and technology on encapsulation.

[36] Speciation hence encompasses so-called *forking* in software as a special case. This is so for two reasons: first, speciation is not always negative as is forking in software, it refers also to, e.g., innovation,

allowing capabilities to go to their most valued use without necessarily resorting to fiat one allows more room for variety. Encapsulation is a way to assure that those who have the capability to perform modifications to programs are assured the rights to do so (cf. **Benkler 2002**; **Garzarelli 2004**). As many have shown (e.g., Langlois 1992; Langlois and Robertson 1995, Ch. 5; Arora et al. 1997; Baldwin and Clark 1997), this is the opposite competitive trend to closed, proprietary systems.

It is thanks to encapsulation that a modular system is able to generate what Garud and Kumaraswamy (1995) have called "economies of substitution": thanks to its free/open nature, a modular system captures not only capabilities that would be present in internal organization, but potentially all those dispersed in the economy. Through economies of substitution it is possible for a number of dispersed organizations or individuals to enjoy cost advantages in production because it is not always necessary to create or search for appropriate capabilities within the organizational boundaries; rather, the capabilities often converge to the production process from outside organizational boundaries through, as it were, their own volition. Alongside the market (Hayek 1945, Chs. 2 and 4; cf. also Raymond 2001), modular organization appears to be the other "distributed environment" (Halpern and Moses 1990) that exploits the comparative advantage of encapsulation to the fullest.

---

improvement, etc.; second, as we just noted, speciation refers not only to final output as does forking. For a definition of forking, visit: ‹**http://people.kldp.org/~eunjea/jargon/?idx=forked**›.

Still, the benefits emerging thanks to the clear definition of property rights have their overhead. That is to say that in attempting to dominate the complexity constraint one is inevitably bound to other constraints.

Modularity decreases *internal coordination costs*. An internal coordination cost is one that emerges when an individual tries to coordinate a number of his or her own activities simultaneously. But thanks to division of labor internal coordination costs decline – e.g., individuals specialize, there are no switching costs from changing tasks, we focus on one task at the time, programmers usually participate to the project they desire, etc. At the same time modularity also creates *external coordination costs*. An external coordination cost is one tied to, e.g., experimentation, learning, and innovation benefits. For example, it refers to the cost of coordinating a large number of specialized programmers simultaneously working on the same patch. But at base, an external coordination costs is an overhead deriving from decreasing internal coordination costs (Houthakker 1994[1956]).

This ineluctable trade-off between internal and external coordination costs originates for cognitive reasons.[37] Hierarchy emerges as a necessity to

[37] In fact, Houthakker (1994[1956], p. 62) uses a parallel with the human brain. "The indivisibility of the individual consists in the fact that, although it may be capable of a great many different activities, it can perform only few activities simultaneously because most activities utilize the same resources and more particularly that coordinating resource which is known as the brain. The larger the number of simultaneous activities, the greater the difficulty of coordinating them and of carrying out each one properly, and the smaller therefore the output from each activity. This applies not only to simultaneous activities, but also to activities that are spread out over time. In the first place some shorter or longer interval is usually needed to switch from one activity to another; in the second place it is usually easier to perform activities that are known from previous experience than to perform them for the first time. All this, the economist will note at once, can be put in terms of increasing returns. We have increasing returns to the extent that, if a single one replaces several activities, there is less need for coordination and

manage the uncertainty generated by economies of substitution. As observed a moment ago, this uncertainty refers not only to speciation in input and output, but also to speciation (and interactions) in the organization of multiple inputs working in parallel, of multiple parallel modular outputs, and of total output. In the case of software some form of management of innovation is necessary to guarantee, e.g., product integrity and modular compatibility.

In view of this, we may interpret hierarchy in voluntary FS/OSS organization as nothing more than the attempt to balance – to search for an equilibrium? – the number of contributors and the number of software contributions accepted. The Debian Constitution, with its clearly defined division of competences, is a concrete illustration of an attempt to create this balance through a formal hierarchy.

More generally, a constitution – designed abstract rules of conduct – is a self-imposed constraint (an external coordination cost) grounded on the belief that there are times when transacting parties have an opportunity to improve their welfare, as well as possibly that of others, by not only making the rules of the game explicit, but also by credibly following them. It is a means to promote specialization because allowing for too much flexibility in action can be equivalent to limiting the ability of others to plan their own purposive action.[38]

---

switching time and more scope for acquiring experience. The output of the single activity may thus be raised above the combined outputs of the several activities."

[38] Compare Buchanan (1999[1990], esp. pp. 380-1), Gifford (1991), Vanberg (1992), and Langlois (1998).

Constitutional constraints are arguably a very effective way to manage routine change, viz., change that does not have a vast impact on existing structure of production. In other words, the incomplete-contract nature of a constitution allows adaptation especially to change that is of a routine type. The dynamic perspective to organization, however, places equal importance on the coordination of change that is not routine. Indeed, organization form – itself not a constant in any moment in time – is *lato sensu* a function of the nature of the change that needs managing.

The most immediate way to get a sense of the argument is probably still in terms of Teece's (1986) distinction between autonomous and systemic innovation. An innovation is said to be autonomous if it is easily adaptable to the existing process of production, viz., if it just requires autonomous changes in one stage of production. In these cases the management of innovation is easier, and thus organization may require little special coordination efforts (where the most extreme example of this would be the market). An innovation is conversely systemic if it requires simultaneous changes in multiple stages of production. The higher uncertainty tied to the coordination of a systemic innovation tips the balance in favor of a more hierarchical/authoritarian organization: centralized direction lowers new transaction costs emerging from the rearrangement of complementary stages of production (Silver 1984; Langlois and Robertson 1995).

One last piece of the puzzle needs to be put in place. It is in situations of systemic change that one is also most likely to witness the hierarchy of

Weberian charismatic authority (Weber 1964[1947]; Coleman 1990) supplanting the one of a constitution. We saw how in the case of Debian there are times when programmers having excellent reputation may intervene to guide the process of production even though they may have no formal organizational title. Why? As is often the case in other situations requiring urgent attention – think, e.g., of a war, epidemic or terrorist attack (cf. Bolton and Farrell 1990) – charismatic authority may be more helpful in overcoming not only multiple, but also high, simultaneous coordination costs. If different external coordination costs all interact at once to solve the same problem they may be all be competing with one another rather than complementing one another. Charismatic authority can obviate these coordination failures through its *super partes* governance structure by aligning conflicting interests. It is a case when programmers commit not to a complex system of rules such as a constitution, but to the direct authority of a charismatic programmer. This is so because constitutional rules may not always allow for the quickest adaptation. Charismatic authority can fundamentally be seen as another way to solve empty core problems emerging in situations of uncommon flux (e.g., reorganizing multiple stages of production, overseeing significant changes in product architecture, etc.).[39]

[39] Langlois (1995, 1998). This is akin to Knight's view of the organizational problem in situations of uncertainty. "When uncertainty is present and the task of deciding what to do and how to do it takes the ascendancy over that of execution, the internal organization of the productive groups is no longer a matter of indifference or a mechanical detail. … Centralization of this deciding and controlling function is imperative, a process of 'cephalization,' such as has taken place in the evolution of organic life, is inevitable, and for the same reasons as in the case of biological evolution" (Knight 1946[1921], III.ix.8, p. 268).

Notice that this versatility interpretation is nothing more than the capabilities approach to organization in another guise. As previously noted, the heterogeneous nature of capabilities is what defines the dynamic organization problem: change along multiple dimensions generated by different capabilities needs frequent coordination. This is one reason why organizations tend to specialize in activities involving *similar* capabilities; but production rests on the *coordination* of activities that are *complementary* (Richardson 1972). In dynamic environments, nonhierarchical organization – such as the market – may fail in transmitting the appropriate knowledge to solve dynamic coordination problems. This is so because the capabilities that need governing contain knowledge that goes beyond price and quantity. Given this, *qualitative* coordination is necessary. In turn, this necessarily entails hierarchy, not only the familiar one of firms, but also the more rare one of voluntary FS/OSS production. Lest the reader may misunderstand, let us underline that this is *not* a plea for planning in the sense of a command economy, but rather one for flexibility (Langlois, e.g., 1995).[40]

[40] In fact, coherence and flexibility are not always strictly in the domain of conscious organization. Consider Hayek's assertion. "There is … no reason why a polycentric order in which each element is guided only by rules and receives no orders from a center should not be capable of bringing about as complex and apparently as 'purposive' an adaptation to circumstances as could be produced in a system where a part is set aside to preform such an order on an analogue or model before it is put into execution by the larger structure. In so far as the self-organizing forces of a structure as a whole lead at once to the right kind of action (or to tentative actions which can be retracted before too much harm is done), such a single-stage order need not be inferior to a hierarchic one in which the whole merely carries out what has first been tried out in a part. Such a non-hierarchic order dispenses with the necessity of first communicating all the information on which its several elements act to a common center and conceivably may make the use of more information possible than could be transmitted to, and digested by, a center" (Hayek 1967, p. 74).

## 6 / Conclusion

This work views modular production through an organizational lens that does *not* hold all *cetera paria*. By focusing on the governance of internal and external coevolutionary coordination costs, through this lens we show that hierarchy (even if perhaps flatter than in most other cases) exists in voluntary FS/OSS production to coordinate an environment subject to frequent mutation.

The substantive lesson is that this more explicitly dynamic reading of the problem of economic organization leads us to argue against what we may christen *modularity determinism*. Modularity determinism may be interpreted as a Panglossian offspring of organizational theories that perceive modularity in input and output as necessarily coinciding with pure modularity (perfect decomposability) in the *process* transforming those inputs into those outputs. To hold, modularity determinism requires the assumption of divisibility of production and exchange (or, what is the same thing, the assumption of no or negligible novelty). We, on the other hand, argue for indivisibility between capabilities and transactions.

To put it differently, a pure modular structure – that is, one lacking hierarchy, such as a market – embeds flexibility in the form of a learning mechanism, but it lacks coherence, the ability to coevolve after adapting to change (cf., e.g., Langlois 1995). Indeed, the flexibility aspect of modularity is what pushes organizational coherence out of synchrony – what misaligns expectations in the Hayekian sense. Hence, when a modular organization

presents coherence (and is healthy), hierarchy, we submit, is bundled to its modularity *because of* flexibility.

Our findings square well with those of others at the level of industry. In their case studies of aircraft engines and chemical plants, Brusoni and Prencipe (2001) for example find that modularity in product architectures is spawning a greater division of labor across firms. At the same time, they write, this increased division of labor requires greater conscious coordination in the form of so-called system integrator companies that must "compose" what has been "decomposed" (p. 195).

Be that as it may, the implications of modularity are still not entirely clear.[41] Pushing our results too far at this stage could mean facing decreasing returns rather quickly. We need further theoretical, empirical (qualitative and quantitative) and historical research.

[41] Compare for example Schilling and Steensma (2001).

## References[42]


Alchian, Armen A. 1977. "Some Economics of Property Rights," in *Idem*, *Economic Forces at Work*. Indianapolis: Liberty Fund, Inc: 127-49. Originally published in 1965, *Il Politico* 30(4): 816-29.

Alchian, Armen A., and Harold Demsetz 1972. "Production, Information Costs, and Economic Organization." *American Economic Review* 62(5): 777-95(December).

Alchian, Armen A., and Susan Woodward 1988. "The Firm is Dead; Long Live the Firm: A Review of Oliver E. Williamson's *The Economic Institutions of Capitalism*." *Journal of Economic Literature* 26(1): 65-79(March).

Alexander, Christopher 1964. *Notes on the Synthesis of Form*. Cambridge, Mass.: Harvard University Press.

Arora Ashish, Alfonso Gambardella, and Enzo Rullani 1997. "Division of Labor and the Locus of Inventive Activity." *Journal of Management and Governance* 1(1): 123-40(November).

Baetjer, Howard Jr. 1998. *Software as Capital. An Economic Perspective on Software Engineering*. Los Alamitos, CA: IEEE Computer Society.

Baldwin, Carliss Y., and Kim B. Clark 1997. "Managing in an Age of Modularity." *Harvard Business Review* 75(5): 84-93(September-October).

Benkler, Yochai 2002. "Coase's Penguin, or, Linux and the Nature of the Firm." *Yale Law Journal* 112(3): 369-446(December). Also online: ‹**http://www.yale.edu/yalelj/112/BenklerWEB.pdf**› (last accessed September 15, 2003).

Bolton, Patrick, and Joseph Farrell 1990. "Decentralization, Duplication, and Delay." *Journal of Political Economy* 98(4): 803-26(August).

Brooks, Frederick P. Jr. 1975. *The Mythical Man-Month: Essays on Software Engineering*. Reading, Mass.: Addison-Wesley.

Brusoni, Stefano, and Andrea Prencipe 2001. "Unpacking the Black Box of Modularity: Technologies, Products and Organizations." *Industrial and Corporate Change* 10(1): 179-205(March).

Buchanan, James M. 1999. "The Domain of Constitutional Economics," in *Idem* (ed.), *The Logical Foundations of Constitutional Liberty. The Collected Works of James M. Buchanan* (Vol. 1). Indianapolis: Liberty Fund, Inc.: 377-95. Originally published in 1990, *Constitutional Political Economy* 1(1): 1-18(Winter).

Buchanan, James M., and Wm. Craig Stubblebine 1962. "Externality." *Economica* (N.S.) 29(116): 371-84(November).

Buchanan, James M., and Yong J. Yoon 2000. "Symmetric Tragedies: Commons and Anticommons." *Journal of Law and Economics* 43(1): 1-15(April).


[42] All web references in the main text not reported here were found to be alive when last checked in November 2003.

Chandler, Alfred D. 1992. "Organizational Capabilities and the Economic History of Industrial Enterprise." *Journal of Economic Perspectives* 6(3): 79-100(Summer).
Coase, Ronald H. 1937. "The Nature of the Firm." *Economica* (N.S.) 4(16): 386-405(November).
Coase, Ronald H. 1960. "The Problem of Social Cost." *Journal of Law and Economics* 3(1): 1-44(October).
Coleman, James S. 1990. "Rational Organization." *Rationality and Society* 2(1): 94-105(January).
Dahlman, Carl J. 1980. *The Open Field System and Beyond. A Property Rights Analysis of an Economic Institution.* Cambridge: Cambridge University Press.
David, Paul 1998. "Common Agency Contracting and the Emergence of 'Open Science' Institutions." *American Economic Review, Papers and Proceedings* 88(2): 15-21(May).
Feller, Joseph, and Brian Fitzgerald 2002. *Understanding Open Source Software Developmen*t. London: Addison-Wesley.
FLOSS Final Report 2002 (June). Online: ‹**http://www.infonomics.nl/FLOSS/report/index.htm**› (last accessed October 20, 2002).
Free Software Foundation, Inc. n.d. *What We Are – Free Software Foundation* (a brochure).
Garud, Raghu, and Arun Kumaraswamy 1995. "Technological and Organizational Designs for Realizing Economies of Substitution." *Strategic Management Journal* 16(Special Issue: Technological Transformation and the New Competitive Landscape): 93-109(Summer).
Garzarelli, Giampaolo 2004. "Open Source Software and the Economics of Organization," in J. Birner and P. Garrouste (eds.), *Markets, Information and Communication*. London and New York: Routledge: 47-62. (A previous version is available online: ‹**http://ideas.repec.org/p/wpa/wuwpio/0304003.html**›.)
Gifford, Adam Jr. 1991. "A Constitutional Interpretation of the Firm." *Public Choice* 68(1-3): 91-106(January).
Halpern, Joseph Y. and Yoram Moses 1990. "Knowledge and Common Knowledge in a Distributed Environment." *Journal of the Association for Computing Machinery* 37(3): 549-87(July).
Hart, Oliver E. 1996. "An Economist's View of Authority." *Rationality and Society* 8(4): 371-86(November).
Hayek, Friedrich A. von 1948. *Individualism and Economic Order.* Chicago: Chicago University Press.
Hayek, Friedrich A. von 1967. *Studies in Philosophy, Politics and Economics.* London and Henley: Routledge & Kegan Paul Ltd.
Helpman, Elhanan, and Manuel Trajtenberg 1998. "Diffusion of General Purpose Technologies," in Helpman, E. (ed.), *General Purpose Technologies and Economic Growth.* Cambridge: MIT Press: 85-119.
Henkel, Joachim 2003. "Open Source Software from Commercial Firms – Tools, Complements, and Collective Invention" (May). Online: ‹**http://www.inno-**

**tec.bwl.uni-muenchen.de/forschung/henkel/OSS_JHenkel_2003-05.pdf**› (last accessed September 16, 2003).

Houthakker, Hendrick S. 1994. "Economics and Biology: Specialization and Speciation," in Buchanan, J. M. and Y. J. Yoon (eds.), *The Return to Increasing Returns*. Ann Arbor: The University of Michigan Press: 61-7. Originally published in 1956, *Kyklos* 9(2): 181-9.

Jensen, Michael C., and William H. Meckling 1992. "Specific and General Knowledge, and Organizational Structure," in W. Lars and H. Wijkander (eds.), *Contract Economics*. Oxford: Basil Blackwell: 251-74.

Klein, Benjamin, Robert G. Crawford, and Armen A. Alchian 1978. "Vertical Integration, Appropriable Rents, and the Competitive Contracting Process." *Journal of Law and Economic*s 21(2): 297-326(October).

Knight, Frank H. 1946. *Risk, Uncertainty and Profit*. London: London School of Economics and Political Science (Re-issue No. 16 in a Series of Reprints of Scarce Tracts in Economics and Political Science; Sixth Impression). First published 1921.

Knight, Frank H. 1967. *The Economic Organization* (*with an article Notes on Cost and Utility*). New York: Augustus M. Kelley Publishers (Reprints of Economic Classics). First edition privately printed in 1933; first published 1951.

Kuan, Jennifer 2001. "Open Source Software as Consumer Integration into Production." Online: ‹**http://opensource.mit.edu/papers/Jenny Kuan - Open Source Software As Integration into Production.pdf**› (last accessed May 2, 2003).

Langlois, Richard N. 1992. "External Economies and Economic Progress: The Case of the Microcomputer Industry." *Business History Review* 66(1): 1-50(Spring).

Langlois, Richard N. 1995. "Do Firms Plan?" *Constitutional Political Economy* 6(3): 247-61.

Langlois, Richard N. 1998. "Personal Capitalism as Charismatic Authority: The Organizational Economics of a Weberian Concept." *Industrial and Corporate Change* 7(1): 195-213(March).

Langlois, Richard N. 2002. "Modularity in Technology and Organization." *Journal of Economic Behavior and Organization* 49(1): 19-37(September).

Langlois, Richard N., and Paul L. Robertson 1995. *Firms, Markets, and Economic Change: A Dynamic Theory of Business Institutions*. London and New York: Routledge.

Lerner, Josh, and Jean Tirole 2002. "Some Simple Economics of Open Source." *Journal of Industrial Economics* 50(2): 197-234(June).

Miller, Mark S., and K. Eric Drexler 1988. "Markets and Computation: Agoric Open Systems," in Huberman, B. A. (ed.), *The Ecology of Computation*. Amsterdam: North-Holland: 133-76. Also online: ‹**http://www.agorics.com/agorpapers.html**› (last accessed March 19, 2002).

Nelson, Richard R., and Sidney G. Winter 1977. "In Search of Useful Theory of Innovation." *Research Policy* 6(1): 36-76(January).

Nelson, Richard R., and Bhaven Sampat 2001. "Making Sense of Institutions as a Factor Shaping Economic Performance." *Journal of Economic Behavior and Organization* 44(1): 31-54(January).

Parnas, David Lorge 1972. "On the Criteria to be Used in Decomposing Systems into Modules." *Communications of the ACM* 15(12): 1053-8(December).

Parnas, David Lorge, Paul C. Clemens, and David M. Weiss 1985. "The Modular Structure of Complex Systems." *IEEE Transactions on Software Engineering* 11(3): 259-66(March).

Peyrache, Eloic, Jacques Crémer, and Jean Tirole 2000. "Some Reflections on Open Source Software." *Communications & Strategies* 40(4): 134-59(Special Issue: From the Net to the New Economy: Critical and Prospective Views).

Radner, Roy 1992. "Hierarchy: The Economics of Managing." *Journal of Economic Literature* 30(3): 1382-415(September).

Raymond, Eric S. 2001. *The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary*, revised edition. Sebastopol, CA: O'Reilly & Associates, Inc. Also online: ‹**http://www.catb.org/~esr/writings/cathedral-bazaar/**› (last accessed September 25, 2003).

Richardson, George B. 1972. "The Organization of Industry." *Economic Journal* 82(327): 883-96(September).

Savage, Deborah A., and Paul L. Robertson 1998. "The Maintenance of Professional Authority: The Case of Physicians and Hospitals in the United States," in P. L. Robertson (ed.), *Authority and Control in Modern Industry: Theoretical and Empirical Perspectives*. London: Routledge: 155-72.

Schilling, Melissa A. and H. Kevin Steensma 2001. "The Use of Modular Organizational Forms: An Industry-level Analysis." *Academy of Management Journal* 44(6): 1149-68(December).

Silver, Morris 1984. *Enterprise and the Scope of the Firm: The Role of Vertical Integration.* Oxford: Martin Robertson.

Simon, Herbert A. 1951. "A Formal Theory of the Employment Relationship." *Econometrica* 19(3): 293-305(July).

Simon, Herbert A. 1998. "The Architecture of Complexity: Hierarchic Systems," in *Idem*, *The Sciences of the Artificial*, 3rd edition, second printing. Cambridge, Mass.: MIT Press: 183-216. Originally published in 1962, *Proceedings of the American Philosophical Society* 106(6): 467-82(December).

Teece, David J. 1986. "Profiting from Technological Innovation: Implications for Integration, Collaboration, Licensing, and Public Policy." *Research Policy* 15(6): 285-305(December).

Torvalds, Linus 1999. "The Linux Edge," in DiBona, Chris, Sam Ockman and Mark Stone (eds.), *Open Sources: Voices from the Open Source Revolution*. Sebastopol, CA: O'Reilly & Associates, Inc.: 101-11. Also online: ‹**http://www.oreilly.com/catalog/opensources/book/toc.html**› (last accessed 21 June, 2003).

Vanberg, Viktor J. 1992. "Organizations as Constitutional Systems." *Constitutional Political Economy* 4(2): 223-53(Spring-Summer).

von Hippel, Eric and Georg von Krogh 2003. "Open Source Software and the 'Private-Collective' Innovation Model: Issues for Organization Science." *Organization Science* 14(2): 209-23(March-April).

Weber, Max 1964. *The Theory of Social and Economic Organization*. Translated by A. M. Henderson and Talcott Parsons. Edited with an "Introduction" by Talcott Parsons. New York: The Free Press. Originally published 1947.

Wheeler, David A. 2003. "Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!" (September 8). Online: ‹**http://www.dwheeler.com/oss_fs_why.html**› (last accessed October 26, 2003).

Wheeler, David A. n.d.1. "Open Source Software/Free Software (OSS/FS) References." Online: ‹**http://www.dwheeler.com/oss_fs_refs.html**› (last accessed January 31, 2002).

Wheeler, David A. n.d.2. "Make your Open Source Software GPL-Compatible. Or Else." Online: ‹**http://www.dwheeler.com/essays/gpl-compatible.html**› (last accessed October 26, 2003).

Williamson, Oliver E. 1991. "Comparative Economic Organization: The Analysis of Discrete Structural Alternatives." *Administrative Science Quarterly* 36(2): 269-96(June).

Williamson, Oliver E. 1999. "Strategy Research: Governance and Competence Perspectives." *Strategic Management Journal* 20(12): 1087-108(December).

Winter, Sidney G. 1982. "An Essay on the Theory of Production," in S. H. Hymans (ed.), *Economics and the World around It*. Ann Arbor: University of Michigan Press: 55-93. A revised version is available online: ‹**http://www.druid.dk/conferences/summer1998/conf-papers/winter.pdf**› (last accessed July 1, 1999).