University of Connecticut

Department of Economics Working Paper Series

Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration

Richard N. Langlois
University of Connecticut

Giampaolo Garzarelli
University of the Witwatersrand

Working Paper 2008-53

April 2008

341 Mansfield Road, Unit 1063
Storrs, CT 06269–1063
Phone: (860) 486–3022
Fax: (860) 486–4463
http://www.econ.uconn.edu/

This working paper is indexed on RePEc, http://repec.org/

Abstract

By employing modularity theory, we study the general phenomenon of open-source collaboration, which includes, e.g., collective invention and open science besides open-source software production. We focus on how open-source collaboration coordinates the division of labor. We find that open-source collaboration is an organizational form based on the exchange of effort rather than of products where suppliers of effort self-identify like suppliers of products in a market rather than accepting assignments like employees in a firm. Our finding suggests that actual open-source software (and other) projects are neither bazaars nor cathedrals, but hybrids manifesting both voluntary production and conscious planning.

Journal of Economic Literature Classification: D02, D23, L17, L23

Keywords: Innovation, Integrality, Intellectual Division of Labor, Modularity, Open Source Software, Theory of the Firm.

Previous versions of this paper have benefited from the feedback received from Davide Consoli, Martin Michlmayr, audiences at DRUID June 18-20, 2006,Copenhagen and EURAM May 16-19 2007, Paris, seminar participants at Wits on April 25, 2007, and three referees of this journal.

Introduction.

We tend to think of decentralized intellectual collaboration as a phenomenon of open-source software, the Internet, and the New Economy. But consider the plight of a certain Monsieur Prony, as recounted by Charles Babbage (1835, §243). Gaspard Riche de Prony (1755-1839) was a French civil engineer whom the Revolutionary government of 1790 had charged with an unenviable assignment: construct the largest and most accurate set of trigonometric and logarithmic tables ever produced. One day, while pondering this seemingly impossible task, Prony wandered into a bookseller's shop and absent-mindedly thumbed through a copy of Adam Smith's *Wealth of Nations*. Suddenly it struck him. He could enlist the division of labor to construct the tables – in effect, he could manufacture logarithms like pins. Driven by this insight, Prony set up a collaborative project along the following lines. He would begin by enlisting four or five of the most eminent mathematicians in France to devise formulas well suited for numerical calculation. The results would then pass to a small team of run-of-the-mill mathematicians who would turn the formulas into simple algorithms. The actual calculations would be performed by a large team of 60 to 80, most of whom knew no math beyond simple addition and subtraction. Indeed, in the event the calculators came largely from the ranks of unemployed hairdressers, a group who had lost their once elaborately coiffed clients to the same Revolutionary taste for austerity and reason that had inspired Prony's

commission. "By 1794, 700 results were being produced each day" (Grattan-Guinness 1990, p. 180).

Obviously, this example differs from present-day open-source software projects in a number of respects. For one thing, Prony's calculators were not volunteers, and were presumably paid either as employees or on a per-calculation basis.[1] Moreover, the entire process was "fordist" rather than collegial: the design was top down, with no communication among the calculators or feedback from them to the mathematicians above. And the calculators were not creative programmers but deskilled automatons whom Babbage had every hope of replacing with his planned difference engine.[2] Nonetheless, like present-day open-source efforts, the Prony Project was an attempt to share-out a complex creative task. Both are examples of what we shall call here the intellectual division of labor.[3]

This paper seeks to understand the phenomenon of open-source collaboration by placing it within this larger context of the intellectual division of labor. In fact, although "open-source" has its original technical meanings in software design and law, we intend to more precisely

---

1 Unfortunately, no information seems to have survived about the actual organization of the work (Grattan-Guinness 1990, p. 179).

2 There is at least one recent analogue to the Prony Project: NASA's Clickworkers study, which enlisted volunteers to engage in well-specified and relatively menial astrometric tasks. See: ‹**http://clickworkers.arc.nasa.gov/top**›. Of course, Babbage's dream of handing off the most menial calculations to computers has long since come true, and today Google will help you donate free time on your Internet-connected computer to solving a small piece of a large scientific problem. See: ‹**http://toolbar.google.com/dc/offerdc.html**›.

3 What Babbage (1835, *passim* Part 3) called the "mental division of labor."

understand the more general open-source production mode. Examples include the phenomenon of "collective invention" (Osteloh and Rota 2004) within industrial communities (Allen 1983) and the professions (von Hippel 1987; Savage 1994); the business of journal editing and refereeing familiar to academics (Bergstrom 2001); online open bibliographic databases, such as Research Papers in Economics (**RePEc**) (Krichel and Zimmermann 2005); cases of "open-source" collaboration for literary and hobbyist ends rather than for software production (like the online encyclopedia **Wikipedia** and the photography site **photo.net**); and even the modern practice of "open science" (David 1998).

Our primary focus is on how open-source collaboration *coordinates* the intellectual division of labor. In Prony's case, the answer was top-down design and control. But the hallmark of many present-day open-source efforts is bottom-up coordination. The Prony Project was fordist; but the paradigmatic open-source effort of today mostly seems to be a horizontal organization where coordination is more often unplanned rather than planned.

To answer the question of how open-source collaborations coordinate the intellectual division we turn to what we may loosely call the modularity theory of the firm (Narduzzo and Rossi 2005; Baldwin and Clark 2006; Langlois 2006), since, as we will argue, open-source collaboration ultimately relies on the institutions of modularity. Our answer hinges on a tradeoff between the benefits (and costs) of modularity and the benefits (and costs) of

its opposite, integrality. This tradeoff determines the extent to which central coordination is preferable to decentralized coordination. In this respect we are attempting to make more precise an idea that is already implicit in many discussions of the open-source phenomenon in software: we try to move a step beyond the many discussions of open-source production that still reason according to the *cathedral* and the *bazaar* polar ideal types (Raymond 2001) by considering the more nuanced characteristics of actual organizational forms.[4] In effect, we argue that the organizational menu is not polar but multidimensional, where forms simultaneously exhibit different degrees of modularity (bazaar) and integrality (cathedral). More specifically, we find that open-source collaboration is a hybrid organizational form that, because of an innate modularity-integrality tradeoff, permits the exchange of effort rather than the exchange of products, and it does so under a regime in which suppliers of effort self-identify like suppliers of products in a market rather than accepting assignments like employees in a firm. We point out, moreover, that it is the nature and intensity of demand – especially the extent to which demand is quality or time sensitive – that can shift the margin between modularity and integrality. And we further suggest that the existence of this tradeoff between modularity and integrality can constitute an incentive and "focal point" for technological or organizational change

[4] For an early statement that the cathedral and the bazaar referred to ideal types rather than to concrete organizational forms, see **Eunice (1998)**.

(Rosenberg 1976) – change that sometimes, and perhaps typically, shifts the margin in favor of modularity.

Modularity and integrality.

In his famous account of the development of the operating system for the IBM 360 series of computers, Frederick Brooks (1975) paints a depressing portrait of the intellectual division of labor. Like Prony, Brooks parceled out the tasks of mental labor to a large number of workers. Unlike Prony, however, Brooks found himself faced with a daunting problem of coordination. An operating system is an immensely complicated tangle of interconnections, and every piece potentially depends on every other piece. Brooks took this to mean that every programmer ought to know what every other programmer is doing, and concluded that coordination costs should rise as the square of the number of workers – an idea now sometimes called Brooks's Law.[5] The implication, he reasoned, is that coordination costs quickly vitiate any benefits from the intellectual division of labor.

At about the same time, however, David Parnas (1972) and other researchers were looking at software systems in a different light. Solving the problem of interdependency, they argued, is not a matter of maximizing

[5] As open-source guru Eric Raymond observes, Brooks's dire conclusion rested on the implicit "assumption that the communication structure of [a] project is necessarily a complete graph" (Raymond 2001, p. 35).

communication among the parts but rather of *minimizing* communication. In this approach, which laid the foundations for object-oriented programming, one attempts to design systems in which not only do the parts not need to communicate extensively with one another but are actually *forbidden* from communicating with one another. The basic idea is that "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" (Parnas *et al*. 1985, p. 260). This is the notion of *information hiding* or *encapsulation*.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_1	x	x	x	x	x	x	x
a_2	x	x	x	x	x	x	x
a_3	x	x	x	x	x	x	x
a_4	x	x	x	x	x	x	x
a_5	x	x	x	x	x	x	x
a_6	x	x	x	x	x	x	x
a_7	x	x	x	x	x	x	x

1. A non-decomposable system.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_1	x	x					
a_2	x	x					
a_3			x	x			
a_4			x	x			
a_5					x	x	
a_6					x	x	
a_7							x

2. A nearly decomposable system

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_1	x	x	x	x	x	x	x
a_2	x	x	x				
a_3	x	x	x				
a_4	x			x	x		
a_5	x			x	x		
a_6	x					x	x
a_7	x					x	x

3. A modular system with common interface.

Figure 1.

Software design reveals the logic of modularity with particular clarity. But the basic ideas were long ago articulated by Herbert Simon (1962) in a more general context. Simon contrasted systems that are *non-decomposable* with systems that are (or are nearly) *decomposable*. Brooks's conception of the 360 operating

system created a non-decomposable system: every part communicates with virtually every other part.[6]

Consider Figure 1, where an entry of x in location a_{ij} means that element a_i communicates with element a_j. Matrix 1 is a fully non-decomposable system: every element communicates with every other element. By contrast, Matrix 2 is a relatively decomposable system. Communication is encapsulated within clusters of elements that do not communicate with elements "far away." Notice, however, that, although a nearly decomposable system like Matrix 2 clearly solves the problem of coordination, it does so by throwing the baby out with the bath water. Matrix 2 is a congeries of autarkic clusters that eliminates the costs of cooperation by the expedient of eliminating cooperation.

The term *modular system* takes on many meanings in the literature; but one important candidate definition, which we adopt here, is that a modular system is a nearly decomposable system that preserves the possibility of cooperation by adopting a common interface. The common interface enables, but also governs and disciplines, the communication among subsystems.[7] In terms of Figure 1, an interface would be a set of elements that communicates with most or all the other elements. In Matrix 3, element a_1 is the common

---

[6] As Baldwin and Clark (2006) rightly insist, the notion of "communication" should involve more than information, and can include the transmission of energy and materials as well as symbols.

[7] This depiction of an interface is only meant to be suggestive. For one thing, cooperation may not require that communication be completely bidirectional in all cases, that is, it

interface: a_1 communicates with all the a_{ij} and all the a_{ij} communicate with a_1.[8] In other respects, however, Matrix 3 remains sparse off the diagonal.

Let us refer to a common interface as *lean* if it enables communication among the subsystems without creating a non-decomposable system, that is, if it enables communication without filling up the off-diagonal. As we will see, an interface may become standardized; it may also be "open" as against "closed." But it is the leanness of the interface, not its standardization or openness, that makes a system modular. Baldwin and Clark (2000) suggest thinking about modularity in terms of a partitioning of information into *visible design rules* and *hidden design parameters*. The visible design rules (or visible information) consist of three parts. (1) An *architecture* specifies what modules will be part of the system and what their functions will be. (2) *Interfaces* describe in detail how the modules will interact, including how they fit together and communicate. And (3) *standards* test a module's conformity to design rules and measure the module's performance relative to other modules. Notice that "standards" in this sense doesn't necessarily imply that the architecture or interfaces are standard in the sense of being publicly shared or common to many similar artifacts or systems. A personal computer, for example, could be modular in Baldwin-and-Clark's sense but

---

may not require that row 1 and column 1 be fully populated. Moreover, many different kinds of interface configurations are possible. On this point see Ulrich (1995).

[8] Think of a_1 as M. Prony's assistant, who must have had to scurry among the calculators, each one working in isolation, to gather up the results and arrange them for typesetting.

still be a unique design incompatible with the architecture and interfaces of other computers.[9]

All in all, then, modularity seems a powerful and elegant solution to the problem of coordinating the intellectual division of labor.[10] No less a figure than Linus Torvalds has testified to the value of modularity in the arena of open-source software.

> With the Linux kernel it became clear very quickly that we want to have a system which is as modular as possible. The open-source 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 (Torvalds 1999, p. 108).

Nonetheless, we argue, there can be costs to modularity as well as benefits, and there can be benefits to non-decomposability – or *integrality*, as we will call it – as well as costs. Indeed, the two are opposite sides of the coin. What modularity does well integrality does poorly, and what integrality does well

---

[9] To put it another way, the term "standards" is sometimes used to mean a set of normative criteria and sometimes to imply replication. Although ultimately related, these are two quite different things. On the various meanings of the term, see David (1987).

[10] Garud *et al*. (2003) collects together many of the foundational articles on modular systems.

modularity does poorly. The costs of the one are essentially the foregone benefits of the other.

The tradeoff between modularity and integrality.

The first kind of benefit from modularity has already occupied us extensively: the ability of a modular system to obviate widespread communication among the modules (or their creators) and to limit unpredictable interactions. In effect, the process of modularization unburdens the system's elements of the task of coordination by handing that function off to the visible design rules. Coordination is imbedded or institutionalized in the structure of the system, which means that it doesn't have to be manufactured on the spot by the participants.

A second source of benefits derives from what Garud and Kumaraswamy (1995) call *economies of substitution*. We can think of these economies of substitution as a species of what economists call economies of scope. Economies of scope exist when it is cheaper to make a given product if you are already making similar products than if you were to start from scratch. This is possible to the extent that you can reuse existing fixed investments (including knowledge) instead of reinventing the wheel. Economies of scope are normally discussed as a property of the "production function" of a firm. But as Langlois and Robertson (1995, p. 5) argue, there can be *external* economies of scope in an open modular system, since the visible design rules constitute a shared fixed investment that everyone can

reuse in creating products through substituting and recombining modules. To the extent that the interfaces governing the modular system are sufficiently standardized, it may be possible to upgrade a system by piecemeal substitution of improved modules without having to redesign the entire system. In large open-source software projects, for instance, the "approach of substituting individual components is the norm."[11] It may also be possible to optimize a system by choosing the best available modules or to customize a system to one's tastes or needs by selecting only some modules and not others[12] (Langlois and Robertson 1992).

A third, and perhaps most important, benefit of modularity is that it militates in favor of specialization and the effective use of local knowledge. If we do not subdivide tasks, everyone must do everything, which means that everyone must know how to do everything. But, as Babbage understood, if we do subdivide tasks, we can assign workers according to comparative advantage. Why pay a mathematician for those parts of the work that an (unemployed) hairdresser could do? More significantly, however, a modular system can do more than use a given allocation of local knowledge effectively – it can potentially tap into a vast supply of local knowledge (Langlois and Robertson 1992). This has not been lost on open-source developers, who often

---

11 Martin Michlmayr, former Debian GNU/Linux Project Leader, personal communication, December 31, 2004. On external scope economies in the case of software, see Baetjer (1998, p. 99).

12 Kuan (2001) and Bessen (2006) see the benefits of open-source software production in terms of its ability to fine tune the product to user needs by making users part of the production process.

wax poetic on the ability of the open-source model to tap into a larger "collective intelligence."[13] Raymond even installs a version of this idea as "Linus's law": "Given enough eyeballs, all bugs are shallow." That is to say: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone" (Raymond 2001, p. 30). A modular system increases the potential number of collaborators; and the larger the number of collaborators working independently, the more the system benefits from rapid trial-and-error learning[14] (Nelson and Winter 1977; Langlois and Robertson 1992; Baldwin and Clark 2000).

Notice here that, unlike the first two kinds of benefits from modularity – institutionalized coordination and economies of substitution – the benefits of tapping into "collective intelligence" depend not only on the technological characteristics of the system itself but also on the way the intellectual division of labor is organized. M. Prony could have set his hairdressers to work under one roof or he could have "put out" the calculations to them in their homes; he could have paid them by the calculation or by the hour. Presumably one set of alternatives would have worked best, and the economics of

---

[13] For example: "The power of the commercial open-source business model is that you're tapping into the collective intelligence of your community." (John Roberts of SugarCRM, quote in LaMonica (2004).)

[14] As Raymond further elaborates, "while coding remains an essentially solitary activity, the really great hacks come from harnessing the attention and brainpower of entire communities. The developer who uses only his or her own brain in a closed project is going to fall behind the developer who knows how to create an open, evolutionary context in which feedback exploring the design space, code contributions, bug-spotting,

organization might shed light on which set that would be. But in all cases, the modularization into simple calculations obviated communication among the calculators and with their handlers. Similarly, IBM could have – and probably did – benefit from economies of substitution in the hardware aspects of the 360 series of computers, even though the system was emphatically closed to outside collaborators, the so-called plug-compatible vendors.[15] But for a modular system to take advantage of extended localized knowledge, the organization of the intellectual division of labor is no longer immaterial. In order to tap into "collective knowledge," the system's interface must be not only lean but also relatively standardized and open.

The economics of networks has taught us that, despite the occasional subtlety, standardization is the easy part. If interfaces are sufficiently lean and sufficiently open, there is a tendency for one of them to emerge as a dominant standard (Shapiro and Varian 1998). Openness is the more interesting issue. As the community of open-source software developers clearly understands, openness does not mean only unfettered access to *knowledge* of the visible design rules of the system, though that may be a necessary condition. Rather, openness is about the *right* to take advantage of

---

and other improvements come from hundreds (perhaps thousands) of people" (Raymond 2001, pp. 50-1).

15 Who famously sued IBM under antitrust laws in an attempt to open access to the system. *United States v. International Business Machines Corporation*, 1956 Trade Cas. #68, 245 (S.D.N.Y. 1956).

those design rules.[16] More broadly, the degree of openness of a modular system is bound up with the overall assignment of *decision rights* within the intellectual division of labor.[17] This is an organizational issue and – what may be the same thing – a *constitutional* issue that is taken up in some detail in the next section.

Let us now consider the other side of the coin: the costs of modularity. The first of these is the (fixed) cost of establishing the visible design rules (Baldwin and Clark 2000). A (nearly) decomposable system may solve coordination problems in an elegant way, but designing such a system may take a considerable amount of time and effort.[18] There may also be costs to communicating the design rules to participants and securing agreement on them.

Another cost is that, at least in principle, it may not be possible to fine-tune a modular system as tightly as an integral one. For many kinds of

---

[16] The question of rights in open-source software is a much-discussed issue, revolving around the various kinds of software licenses possible or in actual use. It would take us too far afield to enter this complex area. But for some distinctions see Wheeler (2007a,b; n.d.).

[17] At this point we have in mind the allocation of decision rights within the intellectual division of labor – for example, the allocation of decision rights to programmers in the process of designing a piece of software. But the same logic applies at the level of the system – *e.g.*, the software – itself. In that case, we can say that the visible design rules themselves are a kind of "constitution" that determines the rights of action that the elements of the system enjoy. And here there is a strong analogy between the idea of encapsulation in software design and private property rights in legal systems (Miller and Drexler 1988; Langlois 2006). In both cases, the operative principle is the creation of a protected sphere of activity permitting autonomy of action. The two levels overlap to the extent that a particular encapsulated subsystem is the property of a particular individual or group within the intellectual division of labor.

[18] Garud and Kumaraswamy (1995) cite evidence that the cost of designing a reusable modular software object may be as much as ten times the cost of designing software intended to be used only once.

software, this may no longer be an important issue in the face of Moore's law.[19] But for other kinds of systems, there may be important performance losses from building a system out of modules. Automobiles, for example, may have an inherent "integrality" that prevents automakers from taking advantage of modularity to the same degree as, say, makers of personal computers (Helper and MacDuffie 2002). One can't swap engines like one swaps hard drives, since a different engine can change the balance, stability, and handling of the entire vehicle. Clayton Christensen and his collaborators (Christensen, Verlinden, and Westerman 2002) have argued that integral designs, which can take advantage of systemic fine-tuning, have an advantage whenever users demand higher performance than existing technology can satisfy. As the fine-tuned system continues to improve in performance, however, it will eventually surpass typical user needs. At that point, these authors argue, production costs move to the fore, and the integral system (and the integrated organization that designed it) will give way to a network of producers taking advantage of the benefits of modularity discussed earlier.

At the same time, however, one might also add that sometimes a modular system can improve in performance even faster than a fine-tuned system. To the extent that such a system benefits from "collective intelligence" and rapid-trial-and-error learning, the improvement in the parts can dominate any benefits from fine-tuning. Personal computers are again a

[19] After Gordon Moore, Intel co-founder, who claimed that the number of functions that can be crammed on a chip doubles every 12 months (Moore 1965).

case in point. PCs have come to outperform first mainframes, then minicomputers, then RISC workstations, all of which, in their day, made their money as fine-tuned non-modular systems (at least relative to PCs). Again, the extent to which modular innovation can outperform fine-tuning may depend on the degree of inherent integrality in the system.

A third, closely related, cost of modularity (benefit of integrality) is the tendency of modular systems to become "locked in" to a particular system decomposition. At least to the extent that knowledge gained creating one modularization of the system cannot be reused in generating a new decomposition, it is a relatively costly matter to engage in systemic change of a modular system since each change requires the payment anew of the fixed cost of setting up visible design rules. If in addition an interface has become a standard, the problems of lock-in are compounded in the way popularized by Paul David (1985), since in that case many people would have simultaneously to pay the fixed cost of change. A modular system is good at modular or *autonomous* innovation, that is, innovation affecting the hidden design parameters of a given modularization but not affecting the visible design rules. But a modular system is bad at *systemic* innovation, that is, innovation that requires simultaneous change in the hidden design parameters and the visible design rules – simultaneous change in the parts and in the way the parts are hooked together.[20]

---

[20] The terms autonomous and systemic are from Teece (1986). There is a third possibility, what Henderson and Clark (1990) call *architectural* innovation. Here the modules remain

The benefits of an integral system in systemic change are related to the benefits of fine-tuning to which Christensen points. Fine-tuning is after all systemic change to improve performance. Thus integral systems may have advantages not only when users demand high performance in a technical sense but also when they need performance in the form of change and adaptability. This latter may also be a function of how *quickly* the user needs the system to perform; the front-end costs of a modular system may take the form of time costs – the output forgone while waiting for the modularization to crystallize or the visible design rules to get worked out. If a modularization is already in place, of course, the system can adapt and respond quickly by simply plugging in new modules to suit user needs.[21] But if there is not yet a modularization, or if the user needs a level of performance greater than can be achieved even with the best possible assortment of available modules, then an integral system may do better.

---

intact, but innovation takes place in the way the modules are hooked together. (For a paradigmatic example of this kind of innovation, visit **Legoland**.) The possibility of architectural innovation underlies the benefits of economies of substitution discussed earlier.

[21] It is for this reason that Figure 3 below grades modular systems as "B" rather than "C" on their ability to fine tune performance or adapt systemically. Modular systems can in fact fine tune and adapt systemically to some degree by taking advantage of economies of substitution.

Spontaneity and design in software production.

Effort and division of labor.

Open-source software production is an organizational form that to a large extent relies on the institutions of modularity driven by exactly the supply side and demand side factors to which Langlois and Robertson (1992) long ago pointed. On the demand side are idiosyncratic user tastes and requirements that call for both high quality and fine degree of customization. On the supply side are the benefits of specialization by comparative advantage and the external economies of a large and diverse talent pool. Consider this description of the **Debian Project**.[22]

> The Debian design process is open to ensure that the system is of the highest quality and that it reflects the needs of the user community. 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

[22] Debian is an association of volunteers who work on an operating system called Debian GNU/Linux. GNU is a recursive acronym that stands for "GNU's Not UNIX"; it is pronounced "guh-NEW" (‹**http://www.gnu.org/**›). Linux refers to the *kernel* – the central functions – of the operating system; GNU/Linux is the complete operating system, including the Linux kernel along with other software components. A *distribution* is simply a (usually complete) packaging of the Linux kernel with other software needed to complete the operating system. Debian GNU/Linux is a GNU/Linux distribution that is produced entirely by volunteers.

> group to anticipate these needs and wants in advance without direct input from others.[23]

But what is the more specific organizational nature of open-source software development efforts?

To answer this question, we present a taxonomy of production models by reasoning along the following lines. Markets are about the exchange of products or outputs; such exchange is coordinated spontaneously, in the sense that relative prices rather than fiat direct resources. A firm stands in contrast to both of these aspects of markets: it replaces contracts for products with employment contracts, effectively substituting a factor market for a product market (Cheung 1983); at the same time, it replaces spontaneous coordination with some kind of central design or direction. Notice that this leaves two unexamined alternatives: product markets governed by central direction and factor markets coordinated spontaneously. Inside contracting and outsourcing are examples of the former. Open-source collaboration is an example of the latter. In the final analysis we find that open-source collaboration manifests both spontaneity (self-selection in terms of which input to contribute) *and* design (conscious direction).

As Jensen and Meckling (1992, p. 251) point out, economic organization must solve two different kinds of problems: "the rights assignment problem (determining who should exercise a decision right), and the control or agency problem (how to ensure that self-interested decision agents exercise their

---

[23] ‹**http://www.debian.org/doc/manuals/project-history/ap-manifesto.en.html**›.

rights in a way that contributes to the organizational objective)." All other things equal, efficiency demands that the appropriate knowledge find its way into the hands of those making decisions. There are basically two ways to ensure such a "collocation" of specific knowledge and decision-making: "One is by moving the knowledge to those with the decision rights; the other is by moving the decision rights to those with the knowledge" (Jensen and Meckling 1992, p. 253). These two choices – as well as possible variants and hybrids – are "constitutions" that set out the assignment of decision rights. Such assignments can take place within firms or within wider networks of independent collaborators.[24]

In the case of the Prony Project, and of fordist production generally, decision rights remain centralized. This is because there is very little knowledge that needs to be transmitted; tasks have been made exceedingly simple, and the important knowledge – that involving design – is already at the center. The agency problem can be addressed either through investments in monitoring or by aligning incentives using a suitable piece rate. Even when the subdivided tasks are far more complicated – and require far more skill and creativity – it is still possible to organize the intellectual division of labor in more-or-less the same way. In this model – which we call *corporate* – the ultimate decision rights remain centralized, even as many de facto decision rights are parceled out to employees at various levels of the hierarchy. Clearly, such an arrangement complicates the agency problem,

[24] On the idea that firms are constitutional systems, see Gifford (1991).

since keeping everyone on the same page is no longer a simple matter of monitoring or incentive alignment in a narrow pecuniary sense.

Many would argue that, even within the corporate context, effective management of high-human-capital projects requires recourse to more "participatory" or collaborative models (Minkler 1993). Does this mean that there is really no difference between the corporate model and more decentralized ones? The answer is no, for two reasons. First, as we have seen, even a large organization is bounded in the capabilities on which it can draw, and this limitation may be important in many cases. Second, the location of the ultimate decision rights matters. For any division of intellectual labor we choose, behavior and performance will be different if we assign decision rights to some central authority rather than to the individual collaborators.[25]

The opposite of a corporate model would be a fully decentralized one in which the collaborators retain the ultimate decision rights. But just as the central holder of decision rights in a corporation must in practice cede de facto decision rights to others, so in a decentralized system the collaborators must give up some pieces of their rights in order to collaborate. In a classic market narrowly understood, the collaborators do this through contract. In

[25] Oliver Williamson (1985, p. 136) traces this effect to the phenomenon of "selective intervention," the tendency of the central holder of decision rights to meddle in the decisions of the collaborators. Henry Hansmann (1996) reminds us that those who possess de facto decision rights will be constrained in their exercise of those rights if they lack the ultimate or "formal" decision rights (which Hansmann equates with ownership). The modern corporation, he points out, is precisely an example in which it pays to assign the formal ownership rights to parties (the stockholders) who may actually be in a poor position to exercise effective control – in part because such an

return for compensation from you, I choose to exercise my right to make shoes by producing the kelly green golf shoes you have contracted for.[26] In the limit, however, I may not even know who "you" are, and I exercise my decision rights in the direction of kelly green golf shoes in the hope that you, or someone like you, comes along. A classic market of this sort is an example of what we call the *spontaneous* model – spontaneous in the sense that the division of labor itself emerges (in the limit) from the choices of the collaborators rather than from a central designer.

This is a perspective on the firm/market dichotomy somewhat different from what one usually encounters in the economics of organization. More typically, one hears the following kind of story: markets are good at exchanging *products* for compensation, whereas firms are good at exchanging *effort* for compensation. The economics of organization can be understood from this perspective as a set of stories about why it is often costly to cooperate by trading products and often necessary to cooperate by trading effort.[27] Ever since Coase (1937), it has been more-or-less taken for granted that the only way to trade effort is through an employment contract: I pay for your time and the right to direct your effort within agreed limits (Simon 1951). In other words, the only way to trade effort is by setting up a firm.

---

allocation keeps the rights out of the hands of other parties (notably the managers) who would abuse them.

[26] Apologies to Deirdre McCloskey (1995).

[27] This is not a denial that effort is ultimately a product. It is merely a claim that effort is a particular kind of product, one whose properties make it costly to trade using only the relatively rudimentary institutional support that an anonymous spot market offers.

Perhaps the most intriguing aspect of the open-source model is that it flies in the face of this assumption: under the right circumstances, it is possible to cooperate spontaneously on the effort margin, not just the product margin.[28] Rather than giving up their decision rights to others, open-source collaborators combine effort "voluntarily." Voluntarily here means not that the collaborators do not receive pecuniary compensation (though that may often be true) but rather that the collaborators choose their own tasks. Assignment of individuals to tasks – and, to an extent we will explore, even the overall design of the division of labor itself – arises from these voluntary choices, in much the same way that assignment of sellers to products in a classic market arises from self-selection.

[28] As pointed out earlier, of course, this model actually antedates software. Notably, it has been the normal mode of organization in the professions (Savage 1994), including those that produce science (David 1998). But only with the prominence of open-source software development has the phenomenon begun to gain the attention of the economics of organization.

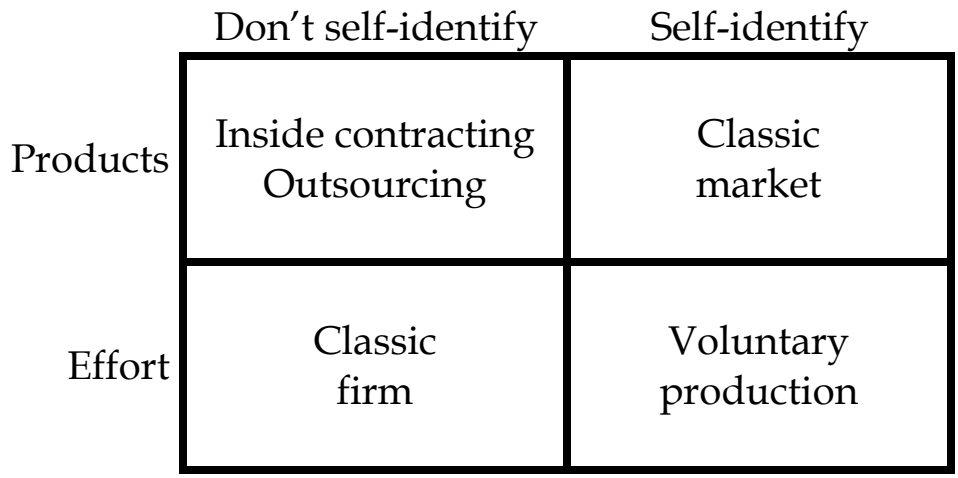

Figure 2

We can distinguish four possibilities. Consider Figure 2. Along the vertical dimension is the issue of design: is assignment to task (and maybe even the division of labor itself) generated through a centralized process or does it arise from the self-identification of collaborators with tasks? Along the horizontal dimension is the problem of information and agency: are we talking about products cleanly measured and priced or are we taking about exchanges of effort that involve costs of measurement and agency? In the upper left-hand box, the division of labor is centrally designed, but the products of that labor are easily measured and priced. This is the world of inside or outside contracting. It is the Prony Project, as well as the outsourcing of intellectual activities like call centers, back-office work, or the reading of X-rays. In the lower left-hand box, the division of labor remains centrally designed, but the cost of measuring and pricing transactions makes it cheaper to purchase the effort of collaborators directly. This is the classic firm. In the upper right-hand box, participants self-select their contributions;

but measurement and pricing costs are not prohibitive, and those contributions take the form of products offered on spec. This is the classic market. Finally, in the lower right-hand box, participants self-select their contributions; but those contributions come directly in the form of effort rather than of effort embodied in a product. That is, we do not have, for instance, a spot market in day labor where day laborers don't choose what they work on; rather, we have a division of labor where programmers do sometimes produce specific products in the end, but in the context of spontaneously adding effort to a larger product not making a product on spec. This is the model of *voluntary* production, the model that most closely resembles the bazaar ideal type.

Rules and division of labor.

This two-dimensional schema has advantages, we argue, over the tripartite distinction Benkler (2002) offers among markets, hierarchies, and what he calls *peer production*. Benkler argues that a perceptible trend toward the increased importance of human capital in production is leading toward peer production and away from both markets and hierarchies. It may well be that, with economic growth and an expanding extent of the market, there is a general trend rightward in Figure 2, what Langlois (2003) calls the phenomenon of the Vanishing Hand. But an increased importance of human capital and greater spontaneity of production is consistent with markets as well as with decentralized collaboration through direct effort. Even apart

from the likes of books, musical scores, or screenplays, there are a plethora of "consulting" services – from legal representation to brain surgery – that are priced on markets.

The most extreme form of a voluntary arrangement would occur when the self-selection of the collaborators itself actually creates the division of labor. This is far from unimaginable: it is exactly what happens in "the market" in the largest sense – including the market for software in the large. It also arguably happens in the context of academic open science, where the pattern of knowledge emerges from the self-selected research choices of the participants. If we cast our gaze down to a less lofty level, however, there almost always seems to be some pre-existing structure of possible tasks from which the participants choose. If Thomas Kuhn (1970) and others are right, even scientific researchers are often – and maybe always – guided in their choice of problem by the constraints of earlier models and approaches. And at the level of any particular software project, the self-selection of workers to tasks takes place within the context of an established architecture or (at the very least) an established technological trajectory.[29]

Why is this so? Consider the experiments conducted by Kevan Davis, a British software engineer (Thompson 2004). Davis set up a website on which visitors could vote, pixel by pixel, on the design of a type font.[30] The

---

29 The idea of a technological trajectory is an application to technology of Kuhn's idea of a scientific paradigm (Dosi 1982).

30 ‹**http://www.typophile.com/**›.

results were quite presentable and, at least when cleaned up a bit, looked "like a mildly punk version of Helvetica, with occasional flashes of creative weirdness, such as the jaunty serif on the foot of the letter 'J'" (Thompson 2004). But when Davis asked people to draw a face or a television set by voting on pixels, the result was a shapeless mess. Efforts at drawing a goat looked mildly goat-like for a while, then collapsed into a jumble after 7,000 votes. The difference between a type font and a goat, of course, is that we come prepared with structural preconceptions that are far tighter in the former case than in the latter. For the font problem, there is something much closer to a pre-existing design architecture to guide individual contributions. This shouldn't be surprising in view of our discussion of modularity. Modularity enables large-scale cooperation; but it requires agreed-upon visible design rules (which, as in the case of letters, may even be tacitly known design rules).

In a typical well-run open-source project, design structure takes the form both of conscious direction and of anonymous rules. It is the first of these elements that makes the development process ultimately a "hybrid" form of organization. Conscious direction is most important in the early and relatively inchoate stages of a project – when design issues remain systemic in important respects. As Ian A. Murdock, creator of Debian, for example recalls:

> Debian's biggest organizational achievement was its emphasis on project management and infrastructure. It's one thing to have a great idea that generates interest; it's another to have the

> necessary infrastructure in place so that, when the masses show up to lend a hand, they can contribute.
>
> In the early days of Debian, the package system and packaging standards we put together ensured that independently developed pieces came together into a cohesive whole. Later, as the number of Debian developers swelled from dozens to hundreds to more than 1,100, a project management infrastructure took shape to handle this massive task.
>
> This, by the way, is why most open-source projects never get beyond the idea phase (just browse through any 20 random projects on SourceForge [‹**http://sourceforge.net/**›] to see what I mean). No matter how great the idea, there has to be a framework in which to contribute, or critical mass can never be reached. And without critical mass, open development projects are not sustainable (**Murdock 2003**).

As a result, Debian volunteers now follow the rules contained in the Debian Social Contract[31], the Debian Free Software Guidelines (aka, DFSG)[32], and the Constitution[33]. These visible design rules assure that "each package can be dropped into the system independently without damaging or interfering with programs from other packages. By working with a set of consistent rules and with identical tools, the volunteers can and do create a truly modular system" (Murdock 1994a). Indeed, over time, design becomes increasingly a matter of impersonal rules, sometimes explicit, sometimes tacit.

But projects almost always retain important elements of conscious direction even as they mature. Like the impersonal design rules, however, these elements of conscious direction involve design at the highest – most

---

31 ‹**http://www.debian.org/social_contract**›.

32 ***Ibidem***. Incidentally, from the DFSG derives the Open Source Definition.

33 ‹**http://www.debian.org/devel/constitution.en.html**›.

abstract – levels of the design hierarchy. Because it takes place at an abstract level, direction in this kind of project plays the role of a (lean) interface among the modular parts rather than of a systemic meddler in an interconnected system.[34] This direction often rests with Project Leaders and core groups of programmers. "The Debian leadership," for instance, "checks to ensure that each package is assembled correctly and that system as a whole is solid; the end result is a set of packages that, though developed and maintained by many different individuals, are as consistent and as professionally constructed as if they were developed by a single person or closely-knit group, but without the limitations imposed by centralized development" (Murdock 1994b).

In terms of our earlier distinction between the corporate model and the spontaneous model, the need for performance and rapid adaptability would tend to militate in the direction of the corporate (Langlois 1988). But, as pointed out, this does not mean that unsatisfied needs for performance and rapid systemic adaptation therefore call for central planning in the socialist sense. In Christiansen's account, unmet performance needs do always call for an integrated corporate structure. But the network theorist Duncan Watts (2004) reminds us that a decentralized structure, with its ability to utilize "collective intelligence," can sometimes be marshaled even in the service of an emergency response. His example is the way the Toyota Corporation

[34] For a more general argument that the degree of abstractness of rules is the fundamental difference between designed and spontaneous orders, see Langlois (1995).

responded in 1997 when the sole plant supplying a crucial component burned to the ground, threatening to bring production of an entire model to a halt. Rather than attempting to create centrally a new plant to make the component, Toyota instead tapped the knowledge and capabilities of a large number of its divisions and outside supplier with the intent of generating rapid trial-and-error learning. "More than 200 companies reorganized themselves and each other to develop at least six entirely different production processes, each using different tools, different engineering approaches, and different organizational arrangements. Virtually every aspect of the recovery effort had to be designed and executed on the fly, with engineers and managers sharing their successes and failures alike across departmental boundaries, and even between firms that in normal times would be direct competitors" (Watts 2004). Within a week, production of the component was back to pre-fire levels.

Clearly, of course, this response was not spontaneous in our sense. It was centrally directed and coordinated to a large extent. But neither was it the standard corporate model. Rather, it is an example of what we call a *hybrid* model – one that has elements both of spontaneous, self-selected

	Modularity	Integrality
Communications costs	A	C
Economies of substitution	A	C
"Collective intelligence"	A	C
Set-up costs	C	A
Fine tuning	B	B
Systemic adaptation	A	B

Figure 3. Grading the alternatives.

production and of central design. In essence, a hybrid model is an attempt to get around the tradeoffs summarized in Figure 3. Such hybrid models are

actually far more typical of open-source software development than are genuinely voluntary or spontaneous ones. Indeed, perhaps *all* models of open-source software development are hybrid models.

Conclusion.

The organizational economics of open-source collaboration (collective invention, the professions, open science, etc.) are not simple. In light of this, we propose that an understanding of such collaboration requires reasoning according to the principles of the emerging economic theory of modularity. These principles allow us to clarify under which circumstances, and in what quantities, central design is necessary, and when a genuinely decentralized design can lead to a well-ordered and effective structure. Such clarification leads us to suggest that open-source collaboration, including the software one, is a hybrid form of organization that presents elements of both the bazaar and cathedral ideal types. More precisely, we argue that open-source collaboration is a type of intellectual division of labor based on the exchange of effort rather than of products where suppliers of effort self-identify like suppliers of products in a market rather than accepting assignments like employees in a firm.

References.

Allen Robert C. 1983. "Collective Invention," *Journal of Economic Behavior and Organization* **4**(1): 1-24 (March).

Babbage, Charles. 1835. *On the Economy of Machinery and Manufactures.* London: Charles Knight, fourth edition. Avaliable at: ‹**http://socserv2.socsci.mcmaster.ca/~econ/ugcm/3ll3/babbage/**›.

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 2000. *Design Rules: the Power of Modularity*. Volume I. Cambridge: MIT Press.

Baldwin, Carliss Y., and Kim B. Clark 2006. "Where Do Transactions Come from? A Network Design Perspective on the Theory of the Firm" Working Paper, Harvard Business School, May 22. Available at: ‹**http://www.people.hbs.edu/cbaldwin/DR2/BaldwinTransactions2006.pdf**›.

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

Bergstrom, Theodore C. 2001. "**Free Labor for Costly Journals?**," *Journal of Economic Perspectives* **15**(3): 183–198 (Summer).

Bessen, James. 2006. "Open-source Software: Free Provision of Complex Public Goods," in Jürgen Bitzer and Philipp J. H. Schröder (eds.), *The Economics of Open Source Software Development*. Amsterdam: Elsevier B. V., Ch. 3. Also available at: ‹**http://www.researchoninnovation.org/opensrc.pdf**›

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

Cheung, Steven N. S. 1983. "The Contractual Nature of the Firm," *Journal of Law and Economics* **26**(1): 1-21 (April).

Christensen, Clayton M., Matt Verlinden, and George Westerman. 2002. "Disruption, Disintegration, and the Dissipation of Differentiability," *Industrial and Corporate Change* **11**(5): 955-993 (November).

Coase, Ronald H. 1937. "The Nature of the Firm," *Economica* (N.S.) **4**(16): 386-405 (November).

David, Paul A. 1985. "**Clio and the Economics of QWERTY**," *American Economic Review*, Papers and Proceedings **75**(2): 332-337 (May).

David, Paul A. 1987. "Some New Standards for the Economics of Standardization in the Information Age," in Partha Dasgupta and Paul Stoneman, eds., *Economic Policy and Technological Performance*. Cambridge: Cambridge University Press, pp. 206-239.

David, Paul A. 1998. "Common Agency Contracting and the Emergence of 'Open Science' Institutions," *American Economic Review*, Papers and Proceedings **88**(2): 15-21 (May).

Dosi, Giovanni. 1982. "Technological Paradigms and Technological Trajectories," *Research Policy* **11**(3): 147-162 (June).

Eunice, Jonathan 1998. *Beyond the Cathedral, Beyond the Bazaar* (May 11). Online: **http://www.illuminata.com/cgi-local/pub.cgi?docid=cathedral**

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).

Garud, Raghu, Arun Kumaraswamy, and Richard N. Langlois, eds. 2003. *Managing in the Modular Age: Architectures, Networks and Organizations*. Oxford: Blackwell Publishing.

Gifford, Adam Jr. 1991. "A Constitutional Interpretation of the Firm." *Public Choice* **68**(1-3): 91-106 (January).

Grattan-Guinness, Ivor. 1990. "Work for the Hairdressers: The Production of de Prony's Logarithmic and Trigonometric Tables," *IEEE Annals of the History of Computing* **12**(3): 177-185 (July-September).

Hansmann, Henry. 1996. *The Ownership of Enterprise*. Cambridge: the Belknap Press of Harvard University Press.

Hayek, Friedrich A. 1948. *Individualism and Economic Order*. Chicago: Chicago University Press.

Helper, Susan, and John Paul MacDuffie. 2002. "B2B and Modes of Exchange: Evolutionary and Transformative Effects," in Bruce Kogut, ed., *The Global Internet Economy*. Cambridge: MIT Press.

Henderson, Rebecca M., and Kim B. Clark. 1990. "Architectural Innovation: the Reconfiguration of Existing Product Technologies and the Failure of Established Firms," *Administrative Science Quarterly* **35**(1): 9-30 (March). Also availbale at: ‹**http://findarticles.com/p/articles/mi_m4035/is_n1_v35/ai_8305916**›.

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, pp. 251-74. Also available at: ‹**http://papers.ssrn.com/sol3/papers.cfm?abstract_id=6658**›.

Krichel, Thomas, and Christian Zimmermann. 2005. "The Economics of Open Bibliographic Data Provision," Working paper 2005-01, University of Connecticut, Storrs, Department of Economics. Available at: ‹**http://ideas.repec.org/p/uct/uconnp/2005-01.html**›.

Kuan, Jennifer W. 2001. "Open-source Software as Consumer Integration into Production," Working Paper (January). Available at: ‹**http://ssrn.com/abstract=259648**›.

Kuhn, Thomas S. 1970. *The Structure of Scientific Revolutions.* Chicago: The University of Chicago Press, second edition.

LaMonica, Martin. 2004. "Breaking the Rules with Open-source**,"** CNET News.com (August 2, 4:00 AM PT) ‹**http://zdnet.com.com/2100-1104_2-5290983.html**›

Langlois, Richard N. 1988. "**Economic Change and the Boundaries of the Firm**," *Journal of Institutional and Theoretical Economics* **144**(4): 635-657 (September).

Langlois, Richard N. 1995. "**Do Firms Plan?**" *Constitutional Political Economy* **6**(3): 247-261.

Langlois, Richard N. 2003. "**The Vanishing Hand: The Changing Dynamics of Industrial Capitalism**," Industrial and Corporate Change **12**(2): 351-385 (April).

Langlois, Richard N. 2006. "**The Secret Life of Mundane Transaction Costs**," *Organization Studies* **27**(9): 1389-1410 (September).

Langlois, Richard N. and Nicolai J. Foss 1999. "**Capabilities and Governance: The Rebirth of Production in the Theory of Economic Organization**," *Kyklos* **52**(2): 201-18.

Langlois, Richard N., and Paul L. Robertson 1992. "**Networks and Innovation in a Modular System: Lessons from the Microcomputer and Stereo Component Industries**," *Research Policy* **21**(4): 297-313 (August).

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

McCloskey, D. N. 1995. "Kelly Green Golf Shoes and the Intellectual Range from M to N," *Eastern Economic Journal* **21**(3): 411-414 (Summer). Also available at: ‹**http://findarticles.com/p/articles/mi_qa3620/is_199507/ai_n8712423**›.

Miller, Mark S., and K. Eric Drexler. 1988. "Markets and Computation: Agoric Open Systems," in Bernardo Huberman (ed.), *The Ecology of Computation*. Amsterdam: North-Holland, pp. 133-176. Also available at: ‹**http://www.agorics.com/agorpapers.html**›.

Minkler, Alanson P. 1993. "The Problem With Dispersed Knowledge: Firms in Theory and Practice," *Kyklos* 46(4): 569-587.

Moore, Gordon. 1965. "Cramming More Components onto Integrated Circuits,"*Electronics* **38**: 114-117 (April 19). Also available at: ‹**http://ieeexplore.ieee.org/iel3/5/14340/00658762.pdf?arnumber=658762**›.

Murdock, Ian. 1994a. "**Overview of the Debian GNU/Linux System**," *Linux Journal* 6es: Article No. 15 (October).

Murdock, Ian 1994b. "**The Open Development of Debian,"** *Linux Journal* 3es: Article No. 7 (June-July).

Murdock, Ian 2003. "Debian: A Brief Retrospective," ‹**http://www.linuxplanet.com/linuxplanet/editorials/4959/1/**›.

Narduzzo, Alessandro, and Alessandro Rossi 2005. "The Role of Modularity in Free/Open Source Software Development," in Stefan Koch (ed.), *Free/Open Software Development*. Hershey, PA: Idea Group Publishing, pp. 84-102.

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

Osterloh, Margit, and Rota, Sandra G. 2004. "Open-source Software Development - Just Another Case of Collective Invention?" Working Paper, University of Zurich (March). Available at: ‹**http://ssrn.com/abstract=561744**›.

Parnas, David Lorge 1972. "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM* **15**(12): 1053-58 (December). Also available at: ‹**http://www.acm.org/classics/may96/**›.

Parnas, David Lorge, Paul C. Clemens, and David M. Weiss 1985. "The Modular Structure of Complex Systems," *IEEE Transactions on Software Engineering* **11**(3): 259-266 (March). Also available at: ‹**http://www.cs.wm.edu/~coppit/other-papers/parnas-modular-structure.pdf**›.

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/**›.

Rosenberg, Nathan 1976. *Perspectives on Technology*. New York: Cambridge University Press.

Savage, Deborah A. 1994. "**The Professions in Theory and History: the Case of Pharmacy**," *Business and Economic History* **23**(2): 130-160 (Winter).

Shapiro, Carl, and Hal R. Varian. 1998. *Information Rules: A Strategic Guide to the Network Economy*. Cambridge: Harvard Business School Press.

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).

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

Smith, Adam 1976. ***An Enquiry into the Nature and Causes of the Wealth of Nations***. Glasgow edition. Oxford: Clarendon Press.

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

Thompson, Clive 2004. "Art Mobs: Can an Online Crowd Create a Poem, a Novel, or a Painting?" *Slate* (July 21). ‹**http://www.slate.com/id/2104087/**›.

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**›.

Ulrich, Karl 1995. "The Role of Product Architecture in the Manufacturing Firm," *Research Policy* **24**(3): 419-440 (May).

von Hippel, Eric 1987. "Cooperation Between Rivals: Informal Know-how Trading,"*Research Policy* **16**(6): 291-302 (December).

Watts, Duncan 2004. "Decentralized Intelligence: What Toyota Can Teach the 9/11 Commission about Intelligence Gathering," *Slate* (August 5). ‹**http://www.slate.com/id/2104808/**›

Wheeler, David A. 2007a. "Why Open-source Software/Free Software (OSS/FS)? Look at the Numbers!" (April 16). Available at: ‹**http://www.dwheeler.com/oss_fs_why.html**›.

Wheeler, David A. 2007b. "Make your Open-source Software GPL-Compatible. Or Else." (June 12). Available at: ‹**http://www.dwheeler.com/essays/gpl-compatible.html**›.

Wheeler, David A. n.d. "Open-source Software/Free Software (OSS/FS) References." Available at: ‹**http://www.dwheeler.com/oss_fs_refs.html**›.

Williamson, Oliver E. 1985. *The Economic Institutions of Capitalism*. New York: the Free Press.