SIEPR Discussion Paper No. 02-27

The Allocation of Software Development Resources In 'Open Source' Production Mode

By
Jean-Michel Dalle
Université Paris VI & IMRI-Université Paris Dauphine
And
Paul M. David
Stanford University
And
Oxford Internet-Institute

March 2003

The Allocation of Software Development Resources in 'Open Source' Production Mode

By

Jean-Michel Dalle
Université Paris VI & IMRI-Université Paris Dauphine
e-mail: jean-michel.dalle@admp6.jussieu.fr

Paul A. David
Stanford University & Oxford Internet-Institute
e-mail: pad@stanford.edu

15 February 2003

["Early Release": see text for a rationale]

"I find that teams can *grow* much more complex entities
in four months than they can *build*."
---Frederick P. Brooks, Jr. *The Mythical Man-Month – 'No Silver Bullet'*

Acknowledgements

We gratefully acknowledge the informative comments and suggestions of Matthijs den Besten and Rishab Ghosh on previous drafts of this paper, as well as Nicholas Carayol's participation in our initial discussions of the modeling approach. Andrew Waterman contributed capable research assistance on a number of critical points in the literature. None of those who have helped can be held responsible for defects that have remained, or for the views expressed here.

This research has drawn support from the Project on the Economic Organization and Viability of Open Source Software, which is funded under National Science Foundation Grant NSF IIS-0112962 to the Stanford Institute for Economic Policy Research. [See: http://siepr.stanford.edu/programs/OpenSoftware_David/OS_Project_Funded_Announcmt.htm]

ABSTRACT

This paper aims to develop a stochastic simulation structure capable of describing the decentralized, micro-level decisions that allocate programming resources both within and among open source/free software (OS/FS) projects, and that thereby generate an array of OS/FS system products each of which possesses particular qualitative attributes. The core or behavioral kernel of simulation tool presented here represents the effects of the reputational reward structure of OS/FS communities (as characterized by Raymond 1998) to be the key mechanism governing the probabilistic allocation of agents' individual contributions among the constituent components of an evolving software system. In this regard, our approach follows the institutional analysis approach associated with studies of academic researchers in "open science" communities. For the purposes of this first step, the focus of the analysis is confined to showing the ways in which the specific norms of the reward system and organizational rules can shape emergent properties of successive releases of code for a given project, such as its range of functions and reliability. The global performance of the OS/FS mode, in matching the functional and other characteristics of the variety of software systems that are produced with the needs of users in various sectors of the economy and polity, obviously, is a matter of considerable importance that will bear upon the long-term viability and growth of this mode of organizing production and distribution. Our larger objective, therefore, is to arrive at a parsimonious characterization of the workings of OS/FS communities engaged across a number of projects, and their collective productive performance in dimensions that are amenable to "social welfare" evaluation. Seeking that goal will pose further new and interesting problems for study, a number of which are identified in the essay's conclusion. Yet, it is argued that that these too will be found to be tractable within the framework provided by refining and elaborating on the core ("proof of concept") model that is presented in this paper.

The Allocation of Software Development Resources in ‘Open Source’ Mode

We aim in this paper to develop a stochastic simulation structure capable of describing the decentralized, micro-level decisions that allocate programming resources both within and among open source/free software (OS/FS) projects, and that thereby generate an array of OS/FS system products each of which possesses particular qualitative attributes. Agent-based modelling of this kind offers a framework for integrating micro-level empirical data about the extent and distribution of participation in “open source” program development, with meso-level observations concerning the social norms and organizational rules governing those activities. It thus takes a step beyond the preoccupation of much of the recent economics literature with the nature of the current and prospective rewards – whether psychic or material – that motivate individuals to develop and freely distribute open source software. Moreover, by facilitating investigation of the “general equilibrium” implications of the micro-behaviors among the participants in OS/FS communities, this modelling approach provides a powerful tool for identifying critical structural relationships and parameters that affect the emergent properties of the macro system.

The core or behavioral kernel of the stochastic simulation model of open source and free software production presented here represents the effects of the reputational reward structure of OS/FS communities (as characterized by Raymond 1998) to be the key mechanism governing the probabilistic allocation of agents’ individual contributions among the constituent components of an evolving software system. In this regard, our approach follows the institutional analysis approach associated with studies of academic researchers in “open science” communities. For the purposes of this first step, the focus of the analysis is confined to showing the ways in which the specific norms of the reward system and organizational rules can shape emergent properties of successive releases of code for a given project, such as its range of functions and reliability. The global performance of the OS/FS mode, in matching the functional and other characteristics of the variety of software systems that are produced with the needs of users in various sectors of the economy and polity, obviously, is a matter of considerable importance that will bear upon the long-term viability and growth of this mode of organizing production and distribution. Our larger objective, therefore, is to arrive at a parsimonious characterization of the workings of OS/FS communities engaged across a number of projects, and their collective productive performance in dimensions that are amenable to "social welfare" evaluation. Seeking that goal will pose further new and interesting problems for study, a number of which are identified in the essay's conclusion. Yet, it is argued that that these too will be found to be tractable within the framework provided by refining and elaborating on the core ("proof of concept") model that is presented in this paper.

A new/old direction for economic research on the phenomenon of OF/FS

The initial contributions to the social science literature addressing the phenomenon of open source and free software (OS/FS hereinafter) have been directed primarily to identifying the motivations underlying the sustained and often intensive engagement of many highly skilled individuals in this non-contractual and unremunerated mode of production.[1] That focus reflects a view that widespread voluntary participation in the creation and free distribution of economically valuable goods is something of an anomaly, at least from the viewpoint of mainstream microeconomic analysis. A second problem that has occupied observers, and especially economists, is to uncover the explanation for the evident success of products of the OS/FS mode in market competition against proprietary software – significantly on the basis not only of their lower cost, but their reputedly superior quality.[2] This quest resembles the first, in reflecting a state of surprise and puzzlement about the apparently greater efficiency that these voluntary, distributed production organizations have been able to attain *vis-à-vis* centrally managed, profit-driven firms that are experienced in creating "closed," software products.

Anomalies are intrinsically captivating for intellectuals of a scientific, or just a puzzle-solving bent. Yet, the research attention that has been stimulated by the rapid rise of an OS/FS segment of the world's software-producing activities during the 1990's owes something also to the belief that this phenomenon and its relationship to the free and open software movements, could turn out to be of considerably broader social and economic significance. There is, indeed, much about these developments that remains far from transparent, and we are sympathetic to the

[1] See, among the salient early contributions to the "economics of open source software," Ghosh (1998), Harhoff, Henkel and von Hippel (2000), Lakhani and von Hippel (2000), Lerner and Tirole (2000), Weber (2000), Kogut and Metiu (2001).

[2] In this particular vein, see, for example Dalle and Jullien (2000, 2003), Bessen (2001), Kuan (2001), Benkler (2002).

view that a deeper understanding of them may carry implications of a more general nature concerning the organization of economic activities in networked digital technology environments. Of course, the same might well be said about other aspects of the workings of modern economies that are no less likely to turn out to be important for human well-being.

Were the intense research interest that OS/FS software production currently attracts to be justified on other grounds, especially as a response to the novelty and mysteriousness of the phenomena, one would need to point out that this too is a less than compelling rationale; the emergence of OS/FS activities at their present scale is hardly so puzzling or aberrant a development as to warrant such attention. Cooperative production of information and knowledge, among members of distributed epistemic communities who do not expect direct remuneration for their efforts simply cannot qualify as a new departure. There are numerous historical precursors and precedents for OS/FS, perhaps most notably in the "invisible colleges" that appeared among the practitioners of the new experimental and mathematically approaches to scientific inquiry in western Europe in the course of the 17th century.[3] The professionalization of scientific research, as is well known, was a comparatively late development, and, as rapidly as it has proceeded, it has not entirely eliminated the contributions of non-professionals in some fields, optical astronomy being especially notable in this regard; communities of "amateur" comet-watchers persist, and their members continue to score – and to verify -- the occasional observational coup.

"Open science," the mode of inquiry that become fully elaborated and institutionalized under systems of public and private patronage during the latter part of the nineteenth and the twentieth centuries, thus offers an obvious cultural and organizational point of reference for observers of contemporary communities of programmers engaged in developing free software and open source software.[4] The "communal" ethos and norms of "the Republic of Science" emphasize the cooperative character of the larger purpose in which individual researchers are

[3] See, e.g., David (1998a, 1998b, 2001c) and references to the history of science literature supplied therein.

[4] This has not gone unrecognized by observers of the free and open software movements. In "The Magic Cauldron," Raymond (1999c) explicitly notices the connection between the information-sharing behavior of academic researchers and the practices of participants in OS/FS projects. Further, Raymond's (1998b) illuminating discussion of the norms and reward systems (which motivate and guide developers selections of projects on which to work) quite clearly parallels the classic approach of Robert K. Merton (1973) and his followers in the sociology of science. This is underscored by Raymond's (1999b) rejoinder to N. Berzoukov's (1999) allegations on the point. See also DiBona et al. (1999) for another early discussion; Kelty (2001), and David, Arora and Steinmueller (2001), expand the comparison with the norms and institutions of open/academic science.

engaged, stressing that the accumulation of reliable knowledge is an essentially social process. The force of its universalist norm is to render entry into scientific work and discourse open to all persons of "competence," while a second key aspect of "openness" is promoted by norms concerning the sharing of knowledge in regard to new findings and the methods whereby they were obtained.

Moreover, a substantial body of analysis by philosophers of science and epistemologists, as well as theoretical and empirical studies in the economics of knowledge, points to the superior efficiency of cooperative knowledge-sharing among peers as a mode of generating additions to the stock of scientifically reliable propositions.[5] In brief, the norm of openness is incentive compatible with a collegiate reputational reward system based upon accepted claims to priority; it also is conducive to individual strategy choices whose collective outcome reduces excess duplication of research efforts, and enlarges the domain of informational complementaries. This brings socially beneficial spill-overs among research programs and abets rapid replication and swift validation of novel discoveries. The advantages of treating new findings as *public goods* in order to promote the faster growth of the stock of knowledge are thus contrasted with the requirement of restricting informational access in order to enlarge the flow of privately appropriable rents from knowledge stocks.

The foregoing functional juxtaposition suggests a logical basis for the existence and perpetuation of institutional and cultural separations between two normatively differentiated communities of research practice. The open "Republic of Science" and the proprietary "Realm of Technology" on this view, constitute distinctive organizational regimes each of which serves a different (and potentially complementary) societal purpose. One might venture farther to point out that the effective fulfilling of their distinctive and mutually supporting purposes was for some time abetted by the ideological reinforcement of a normative separation between the two communities; by the emergence of a distinctive ethos of "independence" and personal disinterested-ness ("purity") that sought to keep scientific inquiry free to the fullest extent possible from the constraints and distorting influences to which commerically-oriented research was held to be subject.

[5] See Dasgupta and David (1994), David (1998c, 2001b) on the cognitive performance of open science networks in comparison with that of proprietary research organizations.

Therefore, if we are seeing something really new and different in the OS/FS phenomenon, that hardly can inhere in attributes shared with long-existing open science communities.[6] Rather, it must be found elsewhere, perhaps in the sheer scale on which these activities are being conducted, in the global dispersion and heterogeneous backgrounds of the participants, in the rapidity of their transactions, and in the pace at which their collective efforts reach fruition. This shift in conceptualization has the effect of turning attention to a constellation of technical conditions whose coalescence has especially affected this field of endeavor. Consider just these three: the distinctive immateriality of "code," the great scope for modularity in the construction of software systems, and the enabling effects of advances in digital (computer-mediated) telecommunications during the past several decades. Although it might be thought that the intention here is merely to portray the historically unprecedented features of the OS/FS movements as primarily an "Internet phenomenon," we have something less glib than that in mind.

It is true that resulting technical characteristics of both the work-product and work-process alone cannot be held to radically distinguish the creation of software from other fields of intellectual and cultural production in the modern world. Nevertheless, they do suggest several respects in which it is misleading to interpret the OS/FS phenomenon simply as "another sub-species of 'open science'." The knowledge incorporated in software differs in at least two significant respects from the codified knowledge typically produced by scientific work groups. Computer software is "technology" (with a small "t"), which it is to say that it becomes effective as a tool immediately, without requiring further expenditures of effort upon development. This immediacy has significant implications not only at the micro-level of individual motivation, but for the dynamics of collective knowledge-production. Indeed, because software code is "a machine implemented as text," its functionality is peculiarly self-exemplifying. Thus, "running code" serves to short-circuit many issues of "authority" and "legitimation" that traditionally have

[6] The phenomenon of free and open source software is perceived by Benkler (2002: pp. 1-2) as an exemplifying "a much broader social-economic phenomenon.…the broad and deep emergence of a new, third mode of production in the digitally networked environment." This mode he labels "'commons-based peer production', to distinguish it from the property- and contract-based modes of firms and markets. Its central characteristic is that groups of individuals successfully collaborate on large scale projects following a diverse cluster of motivational drives and social signals rather than either market prices or managerial commands." Anyone at all familiar with the history of open science since the 17th century will be disconcerted – to say the least --by this particular imputation of novelty and significance to OS/FS projects.

absorbed much of the time and attention of scientific communities; and to radically compress the processes of validating and interpreting new contributions to the stock knowledge.[7]

In our view OS/FS warrants systematic investigation in view of a particular historical conjuncture, indeed a portentous constellation of trends in the modern economy. The first is that information-goods that share these technical properties are moving increasingly to the center of the stage as drivers of economic growth. Secondly, the enabling of peer-to-peer organizations for information distribution and utilization is an increasingly obtrusive consequence of the direction in which digital technologies are advancing. Thirdly, the "open" (and cooperative) mode of organizing the generation of new knowledge has long been recognized to have efficiency properties that are much superior to institutional solutions to the public goods problem which entail the restriction of access to information through secrecy or property rights enforcement. Finally, and of practical significance for those who seek to study it systematically, the OS/FS mode of production itself is generating a wealth of quantitative information about this instantiation of "open epistemic communities." This last development makes OS/FS activities a valuable window through which to study the more generic and fundamental processes that are responsible for its power, as well as the factors that are likely to limit its domain of viability in competition with other modes of organizing economic activities.

Consequently, proceeding from this re-framing of the phenomenon, we are led to a conceptual approach that highlights a broader, ultimately more policy-oriented set of issues than those which have hitherto dominated the emerging economics literature concerning OS/FS. A correspondingly re-oriented research agenda is needed.[8] Its analytical elements are in no way novel, however, but merely newly adapted to suit the subject at hand. It is directed to answering a fundamental and interrelated pair of questions: First, by what mechanisms do OS/FS projects mobilize the human resources, allocate the participants diverse expertise, coordinate the

[7] Therefore, it might well be said that in regard to the sociology and politics of the open source software communities, "the medium is the message."

[8] This is the approach being pursued by the members of the project on The Economic Organization and Viability of Open Source Software at Stanford University and its research partners at academic institutions in France, the Netherlands and Britain. Most of the researchers associated with this project come to this particular subject matter from the perspective formed by their previous and on-going work in "the new economics of science," which has focused attention upon the organization of collaborative inquiry in the "open science" mode, the behavioral norms and reinforcing reward systems that structured the allocation of resources, the relationships of these self-organizing and relatively autonomous epistemic communities with their patrons and sponsors in the public and private sectors. See Dalle, David and Steinmueller (2002) for the scope of this integrated research agenda.

contributions and retain the commitment of their members? Second, how fully do the products of these essentially self-directed efforts meet the long-term needs of software users in the larger society, and not simply provide satisfactions of various kinds for the developers? These will be recognized immediately by economists to be utterly familiar and straight-forward – save for not yet having been explicitly posed or systematically pursued in this context.

Pursuing these questions in more concrete terms brings one immediately to inquire into the workings of the system that actually allocates software development resources among various software systems and applications when the production of code takes place in a distributed community of volunteers, as it does in the OS/FS regime. How does the ensemble of developers collectively "select" among the observed array of projects that are launched, and what processes govern the mobilization of sufficient resource inputs to enable some among those to attain the stage of functionality and reliability that permits their being diffused into wider use – that is to say, use beyond the circle of programmers immediately engaged in the continuing development and 'debugging' of the code itself?

Indeed, it seems only natural to expect that economists would provide an answer to the question of how, in the absence of directly discernible market links between the producing entities and "customers," the output mix of the open source sector of the software industry is determined. Yet, to date, the question does not appear to have attracted any significant research attention. This curious lacuna, moreover, is not a deficiency peculiar to the economics literature, for, it is notable also in the writings of some of the OS/FS movement's pioneering participants and popular exponents.[9] Although enthusiasts have made numerous claims regarding the qualitative superiority of products of the open source mode when these are compared with software systems tools and applications packages developed by managed commercial projects, scarcely any attention is directed to the issue of whether the array of completed OS/FS projects also is "better" or "just as good" in responding to the varied demands of software users.

It is emblematic of this gap that the metaphor of "the bazaar" was chosen by Eric S. Raymond (1998a) to convey the distinctively un-managed, decentralized mode of organization that characterizes open source software development projects – despite the fact that the bazaar

[9] See, e.g., Raymond (1998b, 1999); Stallman (1999), Dibona, Ockman and Stone (1999) and the statements of contributors collected therein.

describes a mode of distribution, not of production. Here is a representative reading of this aspect of Raymond's widely influential essay by an otherwise perceptive commentator, Ko Kuwabara (2000):

> …The Cathedral and the Bazaar, is a metaphorical reference to two fundamentally different styles of software engineering. On the one hand, common in commercial development, is the Cathedral model, characterized by centralized planning enforced from the top and implemented by specialized project teams around structured schedules. Efficiency is the motto of the Cathedral. It is a sober picture of rational organization under linear management, of a tireless watchmaker fitting gears and pins one by one as he has for years and years. On the other hand is the Bazaar model of the Linux project, with its decentralized development driven by the whims of volunteer hackers and little else. In contrast to the serene isolation of the cathedral from the outside, the bazaar is the clamour itself. Anyone is welcome - the more people, the louder the clamour, the better it is. It is a community by the people and for the people, a community for all to share and nurture. It also appears chaotic and unstructured, a community where no one alone is effectively in charge of the community. Not all are heard or noticed, and not all are bound to enjoy the excitement. For others, however, the bazaar continues to bubble with life and opportunity.

Yet, "the bazaar" remains a peculiar metaphor for a system of production: the stalls of actual bazaars typically are retail outlets, passive channels of distribution rather than agencies with direct responsibility for the assortment of commodities that others have made available for them to sell. Given the extensive discussion of the virtues and deficiencies of "the bazaar" metaphor that was stimulated by Raymond's (1998a) essay, it is rather remarkable that the latter's rhetorical finesse of the problem of aligning the activities of producers with the wants of users managed to pass with scarcely any comment.[10]

In contrast, the tasks we have set for ourselves in regard to OS/FS represent an explicit return to the challenge of providing *non-metaphorical* answers to the classic economic questions of whether and how this instance of a decentralized decision resource allocation process could achieve coherent and socially efficient outcomes. What makes this an especially interesting problem, of course, is the possibility of assessing the extent to which institutions of the kind that have emerged in the free software and open source movements are enabling them to accomplish that outcome -- without help either from the "invisible hand" of the market mechanism driven by

[10] See, e.g., Kuwabara (2000), and references in the notes accompanying Raymond (1999: pp.19-63): "Cathedrals and Bazaars."

price signals, or the “visible hands” of centralized managerial hierarchies.[11] Responding to this challenge requires that the analysis be directed towards ultimately providing a means of assessing the social optimality properties of the way “open science”, “open source” and kindred cooperative communities organize the production and regulate the quality of the information-tools and -goods – outputs that will be used not only for their own, internal purposes, but by others with quite different purposes in the society at large.

The general conceptual approach: modelling OS/FS communities at work

The parallels that exist between the phenomena of “open source” and “open science,” to which reference already has been made, suggests a modelling approach that builds on the generic features of non-market social interaction mechanisms. These involve feedback from the cumulative results of individual actions, and thereby are capable of achieving substantial coordination and coherence in the collective performance of the ensemble of distributed agents. This approach points in particular to the potential significance of the actors’ consciousness of being “embedded” in peer reference groups, and therefore to the to role of collegiate recognition and reputational status considerations as a source of systematic influence directing individual efforts of discovery and invention.

Consequently, our agent-based modelling framework has been structured with a view to its suitability for subsequent refinement and use in integrating and assessing the significance of empirical findings -- including those derived from studies of the micro-level incentives and social norms that structure the allocation of software developers’ efforts within particular projects and that govern the “release” and promotion of software code. While it does not attempt to mimic the specific features of collegiate reputational reward systems such as are found in the Republic of Science, it is clear that provision eventually should be made to incorporate functional equivalents of the conventions and institutions governing recognized claims to scientific ‘priority’ (being first), as well as the symbolic and other practices that signify peer approbation of exemplary individual performance.

[11] Benkler (2002) has formulated this problem as one that appears in the organizational space between the hierarchically managed firm and the decentralized competitive market, focuses attention primarily on the efficiency of software project organizations, rather than considering the regime as a whole.

The systems analysis approach familiar in general equilibrium economics tells us, further, that within such a framework we also should be capable of asking how the norms and signals available to micro-level decision-takers in the population of potential participants will shape the distribution of resources among different concurrent projects, and direct the attention of individual and groups to successive projects. That will, in turn, affect the growth and distribution of programmer's experience with the code of specific projects, as well as the capabilities of those who are familiar with the norms and institutions (e.g., software licensing practices) of the OS/FS regime. Obviously, some of those capabilities are generic and thus would provide potential "spill-overs" to other areas of endeavor – including the production of software goods and services by commercial suppliers. From this it follows that to fully understand the dynamics of the OS/FS mode and its interactions with the rest of the information-technology sector one cannot treat the expertise of the software development community as a given and exogenously determined resource. It should be evident from the foregoing that the task upon which we are embarked is no trivial undertaking, and that to bring it to completion we must hope that others can be drawn into contributing to this effort.

We report here on a start towards that goal: the formulation of a highly stylized dynamic model of decentralized, micro-level decisions that shape the allocation of OS/FS programming resources among project tasks, and across distinct projects, thereby generating an evolving array of OS/FS system products, each with its associated qualitative attributes. In such work, it is hardly possible to eschew taking account of what has been discovered about the variety prospective rewards – both material and psychic – that may be motivating individuals to write free and open source software. For, it is only reasonable to suppose that these may influence how they allocate their personal efforts in this sphere. At this stage it is not necessary go into great detail on this matter, but, among the many motives enumerated it is relevant to separate out those involving what might be described as "independent user-implemented innovation."[12] Indeed, this term may well apply to the great mass of identifiably discete projects, because a major consideration driving many individuals who engage in the production of open source would appear to be the direct utility or satisfaction they expect to derive by using their creative

[12] The term evidently derives from von Hippel's (2001, 2002) emphasis on the respects in which open source software exemplifies the larger phenomenon of "user-innovations."

outputs.[13] The power of this motivating force obviously derives from the property of immediate efficiacy, which has been noticed as a distinctive feature of computer programs. But, no less obviously, this force will be most potent where the utilitarian objective does not require developing a large and complex body of code, and so can be achieved quite readily by the exertion of the individual programmer's independent efforts. "Independent" is the operative word here, for it is unlikely that someone writing an obscure driver for a newly-marketed printer that he wishes to use will be at all concerned about the value that would be attached to this achievement by "the OS/FS community." The individuals engaging in this sort of software development may use open source tools and regard themselves as belonging in every way to the free software and open source movements. Nevertheless, it is significant that the question of whether or not their products are to be contributed to the corpus of non-proprietary software, rather than being copyright-protected for purposes of commercial exploitation really is one that they need not address *ex ante*. Being essentially isolated from active collaboration in production, the issue of the disposition of authorship rights can be deferred until the code is written.[14] That is an option which typically is not available for projects that contemplate enlisting the contributions of numerous developers, and for which there are compelling reasons to announce a licensing policy at the outset.

For all intents and purposes software production activity in such circumstances stands apart from the efforts that entail participation in collective developmental process, involving successive releases of code and the cumulative formation of a more complex, multi-function system. We will refer to the latter as OS/FS production in "community-mode" or, for convenience *C-mode,* contrasting it with software production in *I-mode*. Since I-mode products and producers, almost by definition, tend to remain restricted in their individual scope and do not

[13] Just how great a mass of these independent projects represent in the total remains unclear, as the most readily available indications are those obtained by studying the characteristics of the just the *publicly announced* open source projects. On the basis of gathered data from Sourceforge.net on the 100 most active projects observed in the "mature stage" (i.e., the final stage of a project's development, when it is almost fully functional and distributed), Krishnamurthy (2002) reports finding that the modal project has only 1 identified developer; among the most active projects –a mere fraction of the 40 thousand-odd listed on that site -- the median number of developers was 4.

[14] In this respect it can be argued that the decision of the individual developer working in *I-mode* to participate in OS/FS production actually is not a decision about the mode of production, but, instead is a matter of making and *ex post* choice of whether or not to disclose the source code, and whether or not it is worth trying to exploit the resulting program as protected intellectual property. The economics of such post-production decisions certainly are of interest, and the normative force of the open source and free software movements may come into play at this stage. The

provide as direct an experience of social participation, the empirical bases for generalizations about them is still very thin. Too thin, at this point, to support interesting model-building. Consequently, our attention here focuses exclusively upon creating a suitable model to simulate the actions and outcomes of populations of OS/FS agents that are working in *C-mode*.

It would be a mistake, however, to completely conflate the issue of the sources of motivation for human behavior with the separable question of how individuals' awareness of community sentiment, and their receptivity to signals transmitted in social interactions, serves to guide and even constrain their private and public actions; indeed, even to modify their manifest goals. Our stylized representation of the production decisions made by OS/FS developers' therefore does not presuppose that career considerations of "ability signalling," "reputation-building," and the expectations of various material rewards attached thereto, are dominant or even a sufficient *motivations* for individuals who participate in *C-mode* projects. Instead, it embraces the weaker hypothesis that awareness of peer-group norms significantly influences (without completely determining) micro-level choices about the individuals' allocation of their code-writing inputs, whatever assortment of considerations may be motivating their willingness to contribute those efforts.[15]

Our model-building activity aims to provide more specific insights not only into the workings of OS/FS communities, but also into their interaction with organizations engaged in proprietary and "closed mode" software production. It seeks to articulate the interdependences among distinct sub-components of the resource allocation system, and to absorb and integrate empirical findings about micro-level mobilization and allocation of individual developer efforts both among projects, and within projects. Stochastic simulation of such social interaction systems is a powerful tool for identifying critical structural relationships and parameters that affect the emergent properties of the macro system. Among the latter properties, the global performance of

represents a promising line for future research, but it is a line of inquiry quite different from the one we are pursuing here.

[15] It will be seen that the probablistic allocational "rules" derive from a set of distinct community "norms," and it will be quite straightforward within the structure of the model to allow for heterogeneity in the responsiveness to peer-influence in this respect, by providing for inter-individual differences in weighting within the rule-set. This may be done either probabilistically, or by creating a variety of distinct "types" of agents and specifying their relative frequencies in the population from which "contributions" are drawn. For the purposes of the basic model presented here, we have made a bold simplification by specifying that all potential contributors respond uniformly to a common set of allocational rules.

the OS/FS mode in matching the functional distribution and characteristics of the software systems produced to the evolving needs of users in the economy at large, obviously is an issue of importance for our analysis to tackle.

It is our expectation that in this way it will be feasible to analyze some among the problematic tensions that may arise been the performance of a mode of production guided primarily by the internal value systems of the participating producers, and that of a system in which the reward structure is tightly coupled by managerial direction to external signals deriving from the satisfaction of end-users' wants. Where the producers are the end-users, of course, the scope for conflicts of that kind will be greatly circumscribed, as enthusiasts for of "user-directed innovation" have pointed out.[16] But, the latter solution is likely to serve the goal of customisation only by sacrificing some of the efficiencies that derive from producer specialization and division of labour. The analysis developed in this paper is intended to permit investigations of this classic "trade-off" in the sphere of software production.

Behavioral foundations for C-mode production of software

An important point of departure for our work is provided by a penetrating discussion of the operative norms of knowledge production within OS/FS communities, which appears in Eric Raymond's less widely cited essay, "Homesteading the Noosphere" (Raymond, 1999, pp. 65-111).[17] Within the "noosphere" – the "space" of ideas, according to Raymond -- software developers allocate their efforts according to the relative intensity of the reputation rewards that the community attaches to different code-writing "tasks." The core of Raymond's insights is a variant of the collegiate reputational reward system articulated by sociological studies of open science communities: the greater the significance that peers would attach to the project, to the

[16] See von Hippel (2001), Franke and von Hippel (2002), on the development of "user toolkits for innovation," which are specific to a given production system and product or service type, but, within those constraints, enable producers to transfer user need-related aspects of product or service design to the users themselves.

[17] Although Raymond is an astute participant-observer of these OS/FS communities, and his sociological generalizations have the virtue of inherent plausibility, it should be noted that these propositions have yet to be validated by independent empirical tests. Such tests will be provided by analysis of systematic survey or interviews with representative samples of OS/FS community participants, and notably the FLOSS survey and its us counterpart "FLOSS-US" currently undertaken at Stanford University.
See: http://siepr.stanford.edu/programs/OpenSoftware_David/FLOSS-US_announcement.htm

agent's role, and the greater is the extent or technical criticality of his or her contribution, the greater is the "reward" that can be anticipated.

Caricaturing Raymond's more nuanced discussion,[18] we stipulate that (a)launching a new project is usually more rewarding than contributing to an existing one, especially when several contributions have already been made; (b) early releases typically are more rewarding than later versions of project code; (c) there are some rewarding projects within large software system that are systematically accorded more "importance" than others. One way to express this is to say that there is a hierarchy "peer regard," or reputational significance, attached to the constituents elements of a family of projects, such that contributing to the Linux Kernel is deemed a (potentially) more rewarding activity than providing Linux implementation of an existing and widely used applications program, and the latter dominates writing an obscure driver for a newly-marketed printer. To this list we would append an another hypothesized "rule": (d) within each discrete project, analogously, there is hierarchy of peer-regard that corresponds with (and possibly reflects) differences in the structure of meso-level technical dependences among the "modules" or integral "packages" that constitute that project.[19] In other words, we postulate that there is lexicographic ordering of rewards based upon a discrete, technically-based "tree-like" structure formed by the successive addition of project components. Lastly, for present purposes is can be assumed that (e) new projects are created in relation to existing ones, so that always is

[18] See the five, more nuanced "norms" discussed by Raymond (1999: pp.94-97), in a section headed "How Fine a Gift?" We do not enter either here into the related, by distinct discussion that Raymond (1998b, 1999) provides of the important OS/FS community norms regarding recognition and public acknowledgement of "authorship." It is evident, however, that the reputational reward structure must rest upon recognition of "moral possession" of the discrete contributions made to the evolving body of code, just as the reward system of open science communities rests upon citation of scientist's published findings. See, e.g., Dasgupta and David (1994) for an economic recasting of the sociological observations of Robert K. Merton (1973) in this regard. Nor do we follow Raymond (1999: pp. 80-82) in accepting the characterization of the "hacker milieu" as "a Gift Culture," for essentially the same reasons as those which led Dasgupta and David (1987, 1994) to reject that characterization of "open science" communities. Alusions to potlach cermonies among the Kwakiutl of the Pacific Northwest, like references to Mauss on "the Spirit of the Gift" add nothing to an understanding of the nature of the incentives and social mores operating in OS/FS communities, beyond the obvious point that they do not conform to the usual notions of the conduct of agents in a commercial exchange economy.

[19] In the world of open-source software projects, and, indeed, in systematic programming everywhere, production is organized around the building of modules, or packages of code that have a minimal level of "integrity." Files in these packages are linked to, i.e., "call", and/or are "called by" other packages. The package therefore may be conceptualized as situated in a network of "dependency relationships" formed by these links. The intensity of the latter connections can be quantified, for example, in terms of the absolute or relative number of packages that the files forming a module calls (depends upon) and the number of packages that they support (call upon them). As it is feasible to use heuristics in code extraction algorithms to identify which is the supporting, and which the dependent

possible to add a new module in relation to an existing one, to which it adds a new functionality. The contribution made by initiating this new module (being located one level higher in the tree) will be accorded less significance than its counterparts on the structure's lower branches.

Thus, model postulates that the effort-allocation decisions of agent's working in *C-mode* are influenced (*inter alia*) by their perceptions concerning the positioning of the project's packages in a hierarchy of peer-regard; and, further, stipulates that the latter hierarchy is related to the structure of the technical interdependences among the modules.

For present purposes it is not really necessary to specify whether dependent or supporting relationships weigh receive the relatively greater weight in this "calculus of regard." Still, we will proceed on the supposition that modules that are more intensely implicated by links with other packages that include "supportive"connections reasonably are regarded as "germinal" or "stem" sub-routines[20] and therefore may be depicted as occupying positions towards the base of the tree-like architecture of the software project. Assuming that files contributed to the code of the more generic among the modules, such as the kernel or the memory manager of an operating system (e.g., Linux) would be called relatively more frequently by other modules, this might accord them greater "criticality"; or it might convey greater notice to the individual contributor that that which would apply in the case of contributions made to modules having more specialized functions, and whose files were "called" by relatively few other packages. [21]

files, the postulated hierarchy of dependence has an "objective reality" that opens the way for empirical investigation of the conjectured correspondence with perceived hierarchies of "peer-regard."

[20] We have sought here to avoid using "root" to designate the germinal modules, because importing that term from from the arboral metaphor may be confusing for programmers: we are told by one informant that in "Unix-speak" the system administrator is called "root", and the top of the file structure, likewise, is "root." Our hypothesized "dependency tree" might also be in some extent related to the more familiar directory tree structure, but this correlation is likely to very imperfect.

[21] The matter may be illuminated by empirical research that is presently underway to measure the relative frequency of signed and unsigned code across the modules or packages of the Linux Kernel, and to examine whether that frequency measure is correlated with indexes of relative dependency among those packages. (This inquiry is being carried out by Rishab Ghosh in collaboration with Paul David, as part of the Stanford/SIEPR "LICKS" Sub-Project. See: <siepr.stanford.edu/programs/OpenSoftware_David/Licks4b_prop.htm >.) There may be some confounding influences that would similarly impute greater relative importance packages that has particularly intense connections of a supporting form with others. One among them could arise in the ontogeny of the typical computer program. It is true that the files of a programs can be naturally arranged in a hierarchy of dependent links, as functions "calls" sub-functions, which call still other sub-sub-functions. Processes also can be hierarchically arranged so that they appear to spawn sub-processes, etc. On the latter view, it may be the case that -- simply as a consequence of their situation in the temporal sequence of the project's evolution, the initiators of development work (who on other grounds are likely to be accorded greater regard) begin by writing code for modules whose files will spawn (i.e.,support) many other packages. For others who subsequently contribute to further debugging and refinement of the code in those

For the present purposes, Raymond's rules be restated as holding that: (1) there is more "peer regard" to be gained by a contribution made to a new package than by the improvement of existing packages; (2) in any given package, early and radically innovative contributions are more rewarded than later and incremental ones; (3) the lower level and the more generic a package, the more easily a contribution will be noticed, and therefore the more attractive a target it will be for developers. Inasmuch as "contributions" also are acknowledged by Raymond as correcting "bugs of omission", each such contribution – or "fix" – is a patch for a "bug", be it a simple bug, an improvement, or even a seminal contribution to a new package. Therefore every contribution is associated with a variable expected payoff that depends on its nature and "location"[22].

The decision-problem for developers is then to choose which "bug" or "problem" will occupy their attention during any finite work interval. We find here another instance of the classic "problem of problem choice" in science, which the philosopher Charles S. Pierce (1879) was the first to formalise as a microeconomic decision problem. But we need not go back to the static utility calculus of Pierce. Instead, we can draw upon the graph-theoretic model that more has recently been suggested by Carayol and Dalle's (2000) analysis of the way that the successive choices of research agendas by individual scientists can aggregate into collective dynamic patterns of knowledge accumulation. The latter modelling approach is a quite suitable point of departure, precisely because of the resemblance between the reputation game that Raymond (1999) suggests is played by open-source software developers and behavior of open science researchers in response to collegiate reputational reward systems, as described by Dasgupta and David (1994). Although we treat agents' "problem-choices" as being made independently in a decentralised process, they are nonetheless influenced by the context that has been formed by the previous effort-allocating decision of the ensemble of researchers. That context can be represented as the state of the knowledge structure accumulated, in a geological manner, by the "deposition" of past research efforts among a variety of "sites" in the evolving research space – 'the noosphere' of Raymond's metaphor of a "settlement" or "homesteading" process.

modules, the influential consideration may be less a matter of the packages' technical criticality than its offer of greater propinquity (in the code) with the project's "the initiators."

[22] Note that here we neglect, for the moment, the possibility that bugs can become more attractive "targets" because they've existed for long and have thus drawn the attention of the community of developers, and also more specific peer assessments of the "quality" of patches.

A simulation model of OS/FS C-Mode Production

To recapitulate the foregoing exposition, our approach conceptualizes the macro-level outcomes of the software production process carried on by an OS/FS community as a being qualitatively oriented by the interplay of successive individual effort-allocating decisions taken members of a population of developers whose expected behaviors are governed by "norms" or "rules" of the sort described by Raymond.[23] The allocation mechanism, however, is probabilistic rather than deterministic – thereby allowing for the intervention of other influences affecting individual behavior. So far as we are aware, there exist no simple analytical solutions characterising limiting distributions for the knowledge structures that will result from dynamic non-market processes of this kind. That is why we propose to study software production in the open-source mode by numerical methods, using a dynamic stochastic (random-graph) model.

In this initial exploratory model, briefly described, at any given moment a particular OS/FS development "agent" must choose how to allocate a fixed level of development effort -- typically contributing new functionalities, correcting bugs, etc. -- to one or another among the alternative "packages" or modular sub-systems of a particular project. The alternative actions available at every such choice-point also include launching a new module within the project.[24] Agents' actions are probabilistic and conditioned on comparisons of the expected non-pecuniary

[23] We are fully aware of the limits of modelling exercises such as this one. Clearly, it cannot not replicate the world, nor should it attempt to do so. Rather, it may clarify and give insights about the phenomena under examination. Abstracting from the complexity of the actual processes, by means of agent-based models, provides the framework for an interactive program that proceeds abductively – working back and forth interatively between analytical deductions informed by empirical findings, and empirical tests of theoretical propositions. (See Dalle, David and Steinmueller, 2002, for a concrete elaboration of the role played by the stochastical stimulation modelling activity in this larger research program.) Eliciting comments for participant observes in OS/FS projects, especially when it marshals empirical evidence for criticisms of particular abstractions embedded in the simulation structure is there for a vital part of our procedure. It is both a means of improving the usefulness of the simulation experiments performed with the model, and a means of enriching the body of systematic information about processes and structural features of OS/FS organization that experts regard as being especially important. We have made several conscious simplifications in the "reduced-form" formulation presented below, which we flag in the footnotes, and comment upon in the conclusion. But we may also have unknowingly suppressed or distorted other relevant features, and therefore strongly encourage comments on the specifications of the model.

[24] And, in later elaborations of the basic model, launching an entirely different project.

or other rewards associated with each project, given specifications about the distribution of their potential effort endowments.[25]

We consider that open-source developers have different effort endowments, evaluated in thousands of lines of code (KLOC[26]), and normalized according to individual productivities. The shape of the distribution of effort endowments, strictly speaking, cannot be inferred immediately from the (skewed) empirical distribution of the identified contributions measured in lines of code, but one can surmise that the former distribution also is left-skewed – on the basis of the relative sizes of the "high-activity" and "low-activity" segments of the developer population found by various surveys, and notably the FLOSS survey. This feature is in line with the most recent surveys, which have stressed that most open-source contributors engage in this activity on a part-time, unpaid basis.[27] The effort endowment of individuals at each moment in time is therefore given here by an exponential distribution, i.e. smaller efforts will be available for allocation with higher probability. Namely, the density of probability of efforts, denoted by α, is:

$$\alpha = -\frac{1}{\delta}\ln\left(1-p\right), \qquad (1.1)$$

where $p \in \left[0;1\right]$ and δ is a constant. A straightforward calculation will show that here $\left\langle \alpha \right\rangle = \frac{1}{\delta}$, where $\left\langle . \right\rangle$ stands for the means.

Effort endowments measure how many KLOC a given hacker can either add or delete in the existing code, as a common measure of changes of source code in computer science is indeed not only with lines added, but also with lines deleted to account better for the reality of development work, 'bug' correction, and code improvement: therefore it is simply here a question of spending developer time, i.e. writing lines of code, on a given project i.e. module.

[25] In the simplest formulations of the model, agents' endowments are treated as "fixed-effects" and are obtained as random draws from a stationary distribution. More complex schemes envisage endogenously determined and serially correlated code-ing capacities, with allowance for experience based learning effects at the agent level.

[26] "KLOC" is a common measure for such issues in computer science: another implementable unit measurement is the KBOC, which is a more natural metric the data that can be generated by the use of code-extraction algorthims. Function-points provide yet another dimension in which to assess micro-level development contributions, and it therefore may eventually be feasible to treat developer outputs as being vector-valued.

[27] We allow that there may be a small number of participants who are supported, in some cases by commercial employers, to participate in open-source projects on a full time basis.

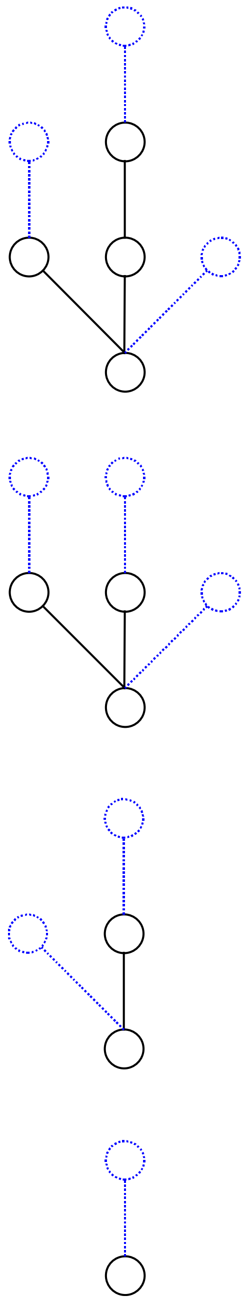

Figure 1. The upwards-evolving tree: a figurative representation of a software system growth process

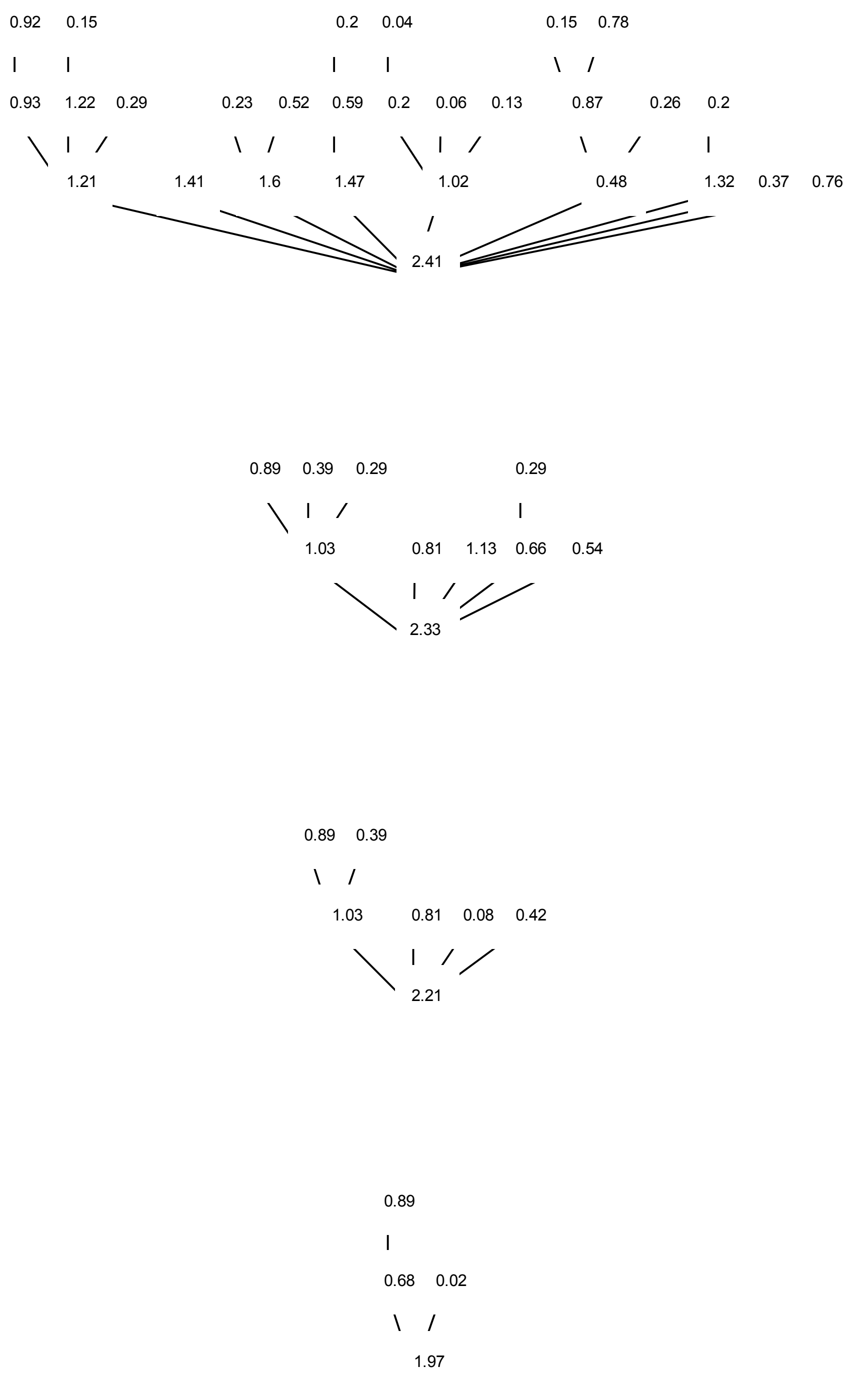

Figure 2. Typical simulation of a software project growth process

Then, as we have argued above, we consider that all the modules, taken together, are organized as in a tree which grows as new contributions are added, and which can grow in various ways depending on which part of it (low or high level modules, notably) developers will select. To simulate the growth of this tree and the creation of new modules, we simply attach a virtual (potential) new node (module, package) to each existing one, at a lower level and starting with version number 0: each virtual module represents an opportunity to launch a new project which can be selected by a developer, and become a real module with a non-zero version number. Figure 1 gives a symbolic representation of the growth process (represented bottom-up) and of the creation of new modules where dashed lines and circles stand for virtual nodes (potential new software packages). Figure 2 present an example of a software tree whose growth (again represented bottom-up) was generated by the stochastic simulation model.

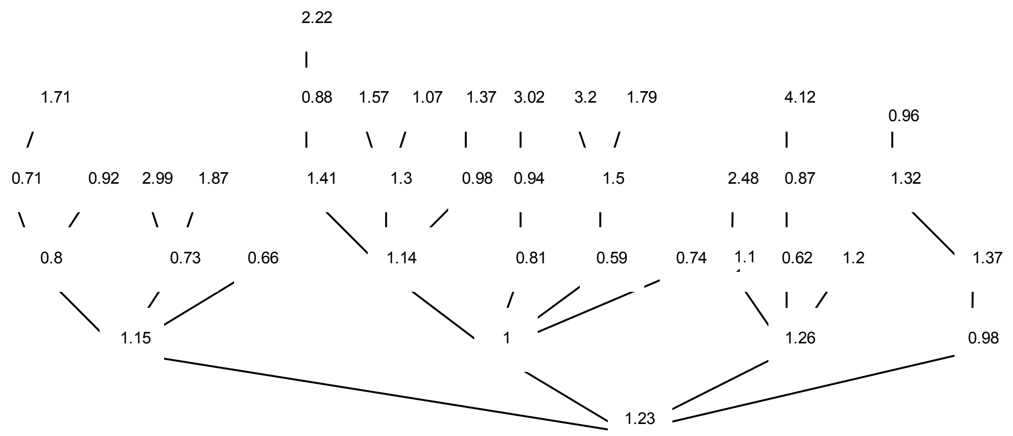

Figure 3. An OS/FS Agent-Generated Software System

We further consider that, for each module, its version number is a good proxy to account for its performance, and that this version number increases non-linearly with the sum of total KLOC added and deleted, here denoted by x, according to:

$$v_d(x) = \log\left(1 + x\, d^{\mu}\right) \quad , \qquad (1.2)$$

where μ is a characteristic exponent and *d* is the distance of the module to the germinal or stem module of the project tree. Further, without loss of generality, we choose the normalization that sets the distance of the stem module itself to be 1. As $d \geq 1$, the specification given by equation (1.2) further implies that it is easier to improve versions for low level modules than for those at higher levels.[28] Figure 3 then represents a typical example of a single tree, where, as in Figure 2 above, numbers associated with each module precisely account for versions.

Then, developers allocate their individual effort endowments at every (random) moment in order to maximise the expected reputation-benefit that it will bring, considering each possible "bug" that is available to be corrected – or each new project to be founded ("bug of omission")[29]. We suppose that the cumulative expected[30] reward (private value) for each existing and potential new project is a function of the version number, and therefore an increasing function of the cumulative efforts measured in KLOC, but also that initial contributions are evaluated as rewarding as long as there are above a given threshold.

$$r_d(x) = v_d(x)\, d^{-\lambda} \qquad (1.3)$$

$$r_d(x) = 0 \; whenever \; v_d(x) \leq v_\theta \qquad (1.4)$$

Here v_θ stands a release "threshold" below which no reward is therefore gained by developers: this threshold accounts for the existence of a norm according to which releasing early is more or less encouraged in OS/FS communities[31]. Namely, it can be rewarding to release

[28] We consider here more or less continuous "release policies" i.e. any improvement in any given module is released as soon as it is contributed. No contribution gets rejected, and accepted contributions are not piled up waiting for a later release. Furthermore, modules are released independently – there is no coordination between the release of several modules, as it is more or less the case when they are grouped into a distribution which gets released regularly, at release dates decided by whoever is in charge of maintaining it. In this first step of our modelling exercise, continuous release stands as an abstraction of Raymond's and others' "release frequently" rule.

[29] To simplify the allocation problem for the purposes of modelling, we consider that a randomly drawn developer, with an associated endowment of effort, makes a commitment to work on a particular "bug" exclusively until that endowment is exhausted. Each "bug" can be assigned an "size," representing the cumulative among of effort that is sufficient to achieve a recognizable "fix" and the model can be formulated in a full information version (allowing developer's to take the "problem size" into account in their allocation decisions), or an incomplete information version (in which developers only know the broad "problem size *range"*), and so cannot see how close to the "solution level" is the cumulative effort that has already been devoted to the bug in question.

[30] This reward is of course actually conditioned by the fact that the project will attract subsequent developers.

[31] This parameter characterizes another aspect of "release policy" norms within the community, as for the "release frequently" rule (see above, note 27).

projects before they are functioning – developers can get "credits" for quite early releases – as it is assumed to be socially efficient because it is a way to attract other developers: an assumption which we will analyse below, and to which we will in fact try to give a better analytical ground.

Note also that in equation (1.3) the reward depends on the height of the project in the software tree – the lower the package, the higher the expected reward, according to a power law of characteristic exponent $\lambda \geq 0$ [32], according to the behavioral foundations of OS/FS community norms as we have abstracted them.

Each existing and potential project is thus associated with an expected payoff depending on its location in the software tree, its current level of improvement (possibly 0) and on individual efforts. More precisely, the expected payoff which corresponds for any given developer to spending its (entire) effort endowment α working on (existing) module m, located at distance d from the root, and whose current level of improvement is x, is:

$$\rho(m) = r_d(x+\alpha) - r_d(x) \qquad (1.5)$$

We suppose that each developer computes the expected rewards associated with each of the nodes according to this last formula and his/her own effort endowment, but also taking into account the rewards associated with the launching of new projects. According to the growth algorithm described above, there is simply one possible new activity -- which would correspond to the creation of a new module -- for each existing package in the global project tree. Numerically, this is strictly analogous to computing the expected reward of "virtual" nodes located as a "son" of each existing node, whose distance to the root module is therefore the distance of the "parent" node plus 1, and whose version and total KLOC are initially 0. Then the expected reward associated with launching a new project as a "son" of node m with effort α is given by:

$$\rho'(m) = r_{d+1}(\alpha) \qquad (1.6)$$

We translate these payoffs into a stochastic "discrete choice" function, considering further that there are non-observable levels of heterogeneity among developers, but that their choice will on average be driven by these expected payoffs. Then:

$$P\left(chosen\ module = module\ m\right) = \frac{\rho(m)}{\sum\limits_{i=1\ (root\ module)}^{number\ of\ modules} \rho(i) + \sum\limits_{i=1\ (virtual\ son\ to\ the\ root\ module)}^{number\ of\ modules} \rho'(i)} \quad (1.7)$$

Our goal then is to examine what pattern of code generation emerges from this system, and how sensitive its morphology (software-tree forms) is to parameter variation, i.e., to variations of the rewards given by the value system of the *OS/FS*-hacker's ethos, and simply to the demography of the population of hackers. The obvious trade-offs of interest are those between intensive effort being allocated to the elaboration of a few "leaves," i.e. modules, which may be supposed to be highly reliable and fully elaborated software systems whose functions in each case are nonetheless quite specific, and the formation of an "dense canopy" containing a number and diversity of "leaves" that, typically, will be less fully developed and less thoroughly "de-bugged".

We therefore focus on social utility measurements according to the basic ideas that:

(1) Low level modules are more valuable than high level ones simply because of the range of other modules and applications that eventually can be built upon them;

(2) A greater diversity of functionalities (breadth of the tree at the lower layers) is more immediately valuable because it provides software solutions to fit a wider array of user needs;

(3) Users value greater reliability, or the absence of bugs, which is likely to increase as more work is done on the code, leading to a higher number of releases. Releases that carry higher version numbers are likely to be regarded as 'better' in this respect[33].

We capture these ideas according to the following simple[34] "social utility" function:

[32] This expected cumulative reward function could also vary depending on the quality of the code, i.e. of its ability to attract early developers or late debuggers, or to grant more reward to all of them.

[33] This formulation treats all bugs symmetrically regardless of where they occur in the code. This is so because the version number of a module that is close to the root is counted the same way as the version of a module that is far from the root. Yet, bugs in low level modules are likely to cause problems for users of many applications than is the case for high level modules that are bug-ridden. This complication could readily be handled by reformulating the social utility measure.

[34] In the future, we might be willing to implement a better differentiation between "functionality" and "reliability", with the idea also that users might typically value both aspects differently from developers.

$$u = \sum_{m}^{(modules)} \left[\left(1 + v_d\left(m\right)\right)^{\nu} - 1 \right] d^{-\xi} \qquad (1.8)$$

Where $\nu \in [0;1]$ and $\xi \geq 0$ are characteristic exponents.

Emergent Properties

Preliminary results[35] tend to stress the social utility of developer community "norms" that accord significantly greater reputational rewards for adding, and contributing to the releases of low level modules. Figure 4 presents the typical evolution of social utility with λ (efficiencies are averaged over 10 simulation runs, while other parameters remain similar i.e. $\delta = 3 \quad \mu = 0.5 \quad \nu = 0.5 \quad \xi = 2$)[36].

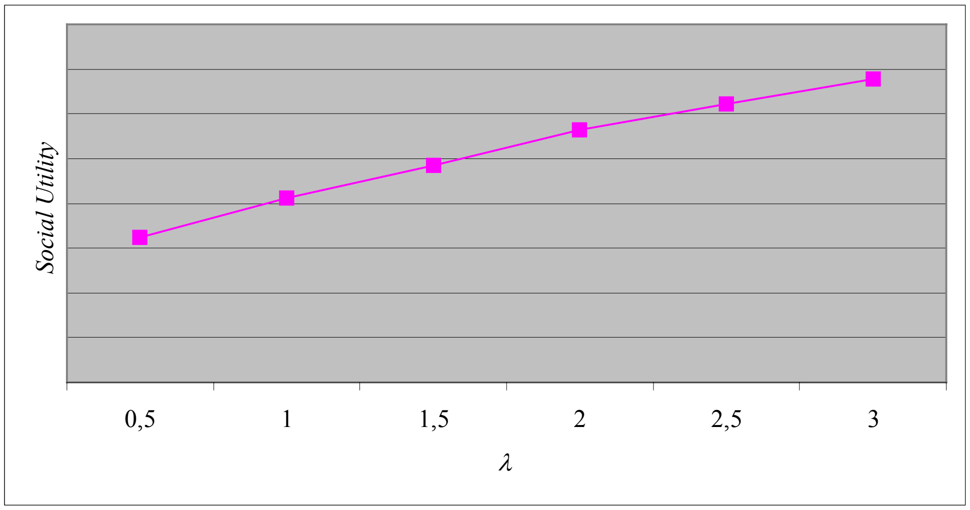

Figure 4

[35] This is based upon a static ex post evaluation of the resulting tree form, and it is evident that the results may be altered by considering the dynamics and applying social time discount rates to applications that only become available for end users at considerably later dates. In other words, the social efficiency of the reward structure that allocates developers' efforts will depend upon the temporal distribution, as well as relative extent to which OS/FS-generated code meets the needs of final users rather than the needs/goals of the agents who choose to work on these projects.

[36] This result holds for various other values of these parameters, although more complete simulations are needed to assess the range of its validity. To exclude a potential artifact, note that this result also holds if new nodes are created at the same distance from the root as their 'parent' node (instead of their parent node's distance plus one).

Further, our preliminary explorations of the model suggest that policies of releasing code "early" tend to generate tree-shapes that have higher social utility scores. Then, Figure 5 gives the evolution of social utility depending on v_θ (here, utilities are averaged over simply 5 simulation runs, while $\delta = 3 \quad \mu = 0.5 \quad \nu = 0.5 \quad \xi = 2 \quad \lambda = 2$)[37].

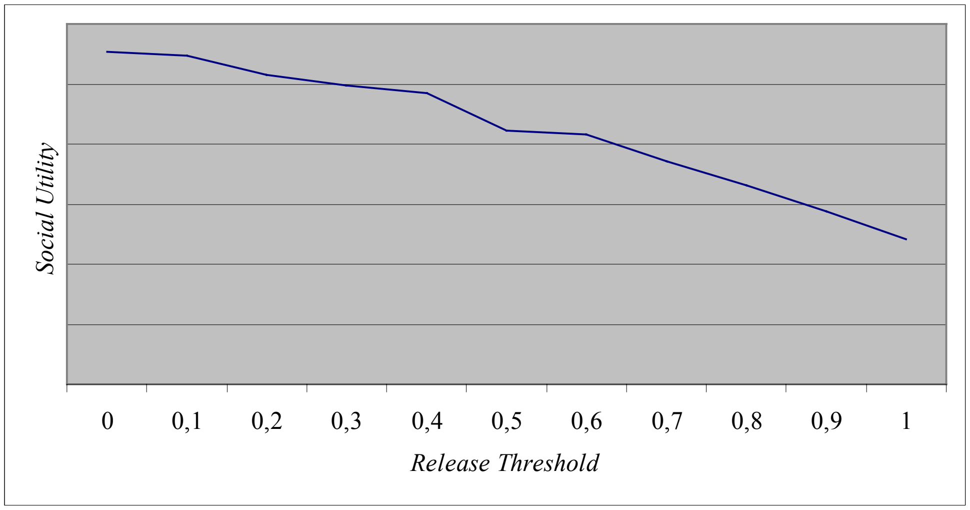

Figure 5

The intuitively plausible interpretation of this last finding is that early releases create bases for further development, and are especially important in the case of low level modules as they add larger increments to social utility. The reputational reward structure posited in the model encourages this "roundabout" process of development by inducing individual efforts to share the recognition for contributing to code, and notably to low level code. Figure 6 brings some rather conclusive evidence in favor of this explanation by displaying the number of modules at "level 2", i.e. at distance 1 from the kernel ("geminal" or "stem") module.

[37] This result holds for various other values of these parameters, although more complete simulations are needed to fully assess the range of its validity.

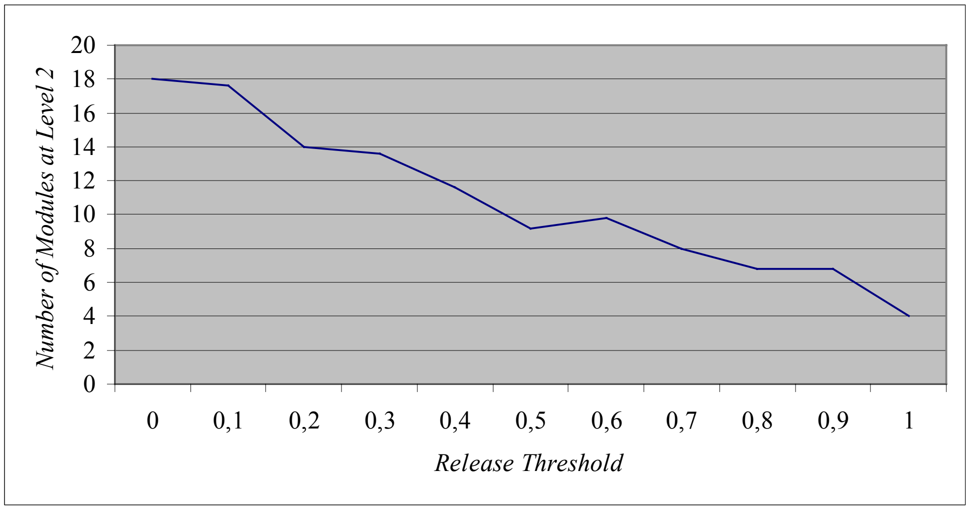

Figure 6

To go one step further, we suggest that early releases of low level modules could be considered as *seminal*, according to an expression often used to characterize important and initial scientific contributions (articles), meaning that these contributions, however limited, create subsequent and sufficient opportunities for other developers to earn reward by building on them.

This points to the functional significance of one of the strategic rules – "release early" and "treat your users as co-developers" – that E.S. Raymond has put forward for open-source development, in the classic exposition, 'The Cathedral and the Bazaar'. As Raymond himself puts it:

> [Treating your users[38] as co-developers] The power of this effect is easy to underestimate …. In fact, I think Linus [Torvalds]'s cleverest and most consequential hack was not the construction of the Linux kernel itself, but rather his invention of the Linux development model. When I expressed this opinion in his presence once, he smiled and quietly repeated something he has often said:

[38] The fact that these co-developers are users essentially guarantees that they provide solutions to existing and relevant problems: this is related to von Hippel's analysis of OSS as "user-innovation", but also to another of Raymond's observations, according to which only contributions are useful in open-source development, as opposed to people showing up and proposing to "do something". Furthermore, this is close from the "given enough eyeballs, all bugs are shallow" rule, and from one of the key reasons why open-source development (Linus's Law) appears to violate Brooks' Law: "Adding more programmers to a late project makes it later.", and although the citation we have put in front of this paper tends to proved that, in a later work, Fred Brooks had the intuition that software productivity could actually be improved if software was grown instead of built. Here, the "release early" and "attract users-co-developers" rules stand as necessary conditions for this property to hold because they make the set of existing problems explicit to all those who might be able not only to encounter them, as users, but still more importantly so to solve them, as co-developers, while be rewarded in doing so and increasing also the author of the seminal contribution's final reward.

"I'm basically a very lazy person who likes to get credit for things other people actually do."

By this, we can understand the mechanism for eliciting seminal contributions, i.e., of early release and attraction of co-developers, to operate in the following way: rewarding early release, and allowing others to build upon it, does not simply create a sufficiently rewarding opportunity for potential co-developers to be attracted, but also brings extra reward to the individual who has disclosed a seminal work. Here, at least for low level modules, interdependent expected rewards are such that they create incentives for what Raymond (1999: p. 27) calls "loosely-coupled collaborations enabled by the Internet" – that is to say, for cooperation in a positive-sum game, positive both for the players and for social efficiency In a sense, and at a meta-level, Linus Torvalds's seminal contribution was not only the kernel, but a new method of software development, which was indeed new and different from the more classical methods which had previously been supported by the FSF for most GNU tools (ibidem, pp. 27 & 29). Once again:

> Linus (Torvalds) is not (or at least, not yet) an innovative genius of design in the way that, say, Richard Stallman or James Gosling (of NeWS and Java) are. Rather, Linus seems to me to be a genius of engineering and implementation, with ….a true knack for finding the minimum-effort path from point A to point B. … Linus was keeping his hackers/users constantly stimulated and rewarded – stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant (even *daily*) improvement in their work.

The price to be paid for implementing such an early release scheme is of course that the higher number of modules being created come at the sacrifice lower level versions that might have been produced with equivalent levels of efforts. Figure 7 presents the evolution of the version of the kernel, of the average version of level 2 modules, and of the average version of the modules over the entire software tree depending on the release threshold v_θ (same parameter values, still averaged over 5 simulation runs).

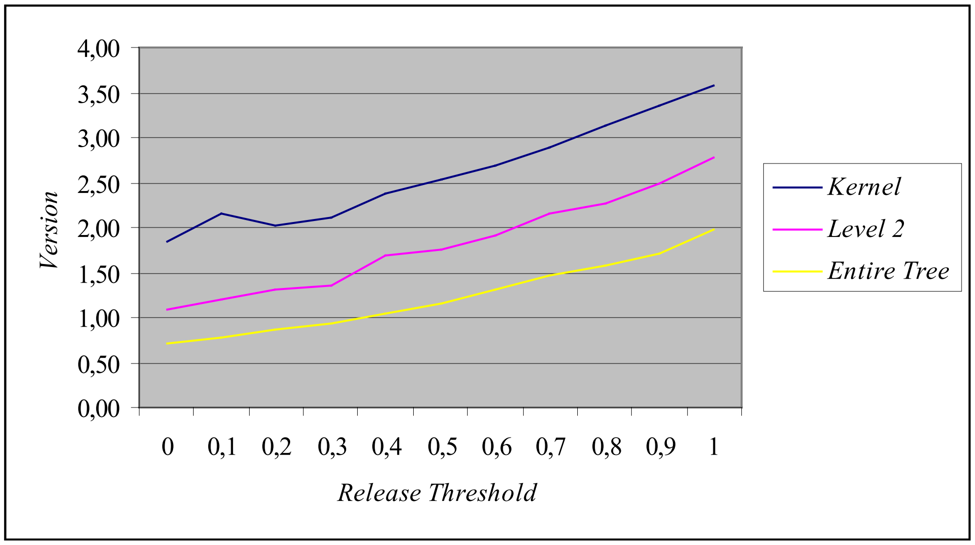

Figure 7

Conclusion, and To-Do List

Although there are clearly many things to be improved in this very preliminary attempt to model the workings of OS/FS communities, we hope that we might have brought some preliminary evidence about what kind of insights such a tool might provide to observers and practitioners of OS/FS. And, as it has provided a rationale for "early release" policies, we have opted for an early release of our work on this topic: in a sense, one should look at all the footnotes in the former sections not only as disclaimers about the preliminary nature of our attempts, but also as opportunities for improvement and for co-development of our model. Although we certainly "commit" to do part of this job, we are also convinced that, considering the complexity of OS/FS communities, we need to harness significant effort to develop a proper model of OS/FS communities.

In this respect, and standing as a temporary conclusion, let us briefly summarize for now at least part of the "to-do" list of features that should be added to the model:

(i) *Microbehaviors*. Clearly, the behavior of developers (contributors) thus far is caricatured as myopic and, more seriously, still lacks several important dynamic dimensions. Learning is missing, for instance: as a matter of fact, acquiring the skills to debug a particular module, or to add new functionalities to it, is not costless. But the model does not make allowance for these "start-up" costs, which would affect decisions to shift attention to a new package of code in the project. Secondly, instead of choosing how to apply their currently available "flow" development inputs ("code-writing time, in efficiency units) among alternative "modules," developers might consider "aggregating" their efforts by working "off-line"over of longer interval. Intertemporal investment strategies of this sort would permit individuals to make a larger, and possibly more significant contribution to a module, and thereby garner greater peer-recognition and rewards.[39] Thirdly, and perhaps most obviously, the model in its presently simplified form abstracts entirely from behavioral heterogeneities. The latter could derive from the variety of motivations affecting the effort that developers are willing to devote to the community project, or to differences in preferences for writing code, as distinct from engaging in newsgroup discussions with other contributors. But, as we have modelled effort in efficiency units (KBOCs per period), differences in innate or acquired skill among contributors also would contribute to generating a (changing) distribution of input capacities in the developer population. The convolution of that distribution with the distribution of motivational intensities would then have to be considered by the simulation model when a "potential developer" is drawn at random from the population, for inter-individual differences in the extent of the (effective) "endowment" would influence the (simulated) pattern of micro-behaviors.

(ii) *Release policies*, can be viewed as reflecting the governance structure of a project and therefore treated as a "pre-determined" variable, or "fixed effect" that potentially distinguishes one project from another.[40] Such policies can be viewed as a factor influencing the distribution of developer efforts among different OS/FS projects, and thereby affecting their relative advance toward "maturity." But, as differences among the operating rules followed by maintainers of different modules within a complex

[39] What makes this an interesting strategic decision to model is the risk that while working "off line," so to speak, for an extended period, and not submitting increments of code in more continuous flow, someone else might submit a discrete contribution that would have the same functional attributes, thereby preempting the investment's chance of being accepted. The perceived hazard rates for "credit losses" of that sort might be modelled as rise as more developers gain familiarity with a given module, or others technically related to it.

[40] Such policies can be treated as applying uniformly across all the modules of a project, or as defining a pre-specified range of release intervals defined either in temporal terms, or in terms of incremental code.

project would create *de facto* local policy variations release rules, this too can be incorporated by the model among the set of conditions affecting the internal allocation of developers' contributions. Global release policies, affecting by how accessible the project's code is to users through one or more, for-profit and non-profit "distributions"of its code, constitutes yet another important aspect of performance. This may affect both perceived reliability, market adoption, and so feed back to influence the project's success in mobilizing supporting resources both within the developer community and from external sources.

(iii) *Willingness to contribute to different projects*. As has been noted, developers might have variable effort endowments, depending for instance on the global shape of a project, or on other variables such as its market share, release policies, licensing schemes, etc. This will affect choices among different projects. But the positioning the of projects in the "software systems product space," and in particular their relationship to projects that are at work on product substitutes, is another aspect of the dynamics of resource allocation in the developer community at large. It will therefore important to extend the model in this direction, by defining the dimensions of the "product space"; only when "categories can be represented" will it become possible to simulate the effects of what Raymond (1999) describes as "category killers" – project trees, in our metaphor, that block the sunlight and absorb the nutrients in the area around them, preventing other project-trees from establishing themselves there

(iv) *Users*. End-users have not really been implemented yet in the model, save for the fact that developers are assumed to be also users in that they know what the bugs (actual ones, and bugs of omission) are! Users are likely, as a group, to have different preferences from developers, for instance, being disposed to grant more weight to reliability rather than to the range of functionalities embedded in a single program. Furthermore, some developers (some communities?) may be more strongly motivated than others to work on "popular" projects, that is, by projects which are able to attract users from the general, in expert population by fulfilling their working requirement, affording network compatibilities with co-workers, being properly distributed.[41]

[41] Indeed, there may we some developers who would be quite insensible to those motivations, even shun projects of that kind, believing that commercial software vendors would cater to those needs, and that they would serve the

Again, it would be appropriate for the model to represent such considerations and, by allowing for alternative distributions of developer attitudes, investigate their potential impacts upon the pattern of OS/FS project development.

(v) *Sponsorship*, and more generally symbiotic relationships with commercial entities of various kind (ancillary service companies, editors of complementary commercial application packages, even proprietary software vendors, etc.), can influence OS/FS development by adding and directing efforts. This can take a variety of forms, ranging from commercial distribution of OS/FS-based products to hiring prominent developers and letting them contribute freely to selected open-source projects. The interaction with complementary enterprises in the software systems and services sector, therefore, will have to be modelled along with the direct competition between the underlying OS/FS code and the products of commercial vendors of proprietary software and bundled services.

(vi) *Authority and Hierarchies:* In a sense, the reputation rewards associated with contributing to the development of a project are only obtained if the developers submitted "patches" are accepted by the module or project maintainer. Rather than treating the latter's decisions as following simple "gate-keeping" (and "bit-keeping') rules that are neutral in regard to the identities and characteristics of the individual contributors, it may be important to model the acceptance rate as variable and "discriminating" on the basis the contributing individuals' "experience" or "track-records." This would enable the model to capture some features of the process of "legitimate peripheral participation" through which developers are recruited. Modules towards the upper levels in the tree, having fewer modules calling them, might be represented as requiring less experience for a given likelihood of acceptance. Comparative neophytes to the OS/FS community ("Newbies") thus would have incentives to start new modules or contribute to existing ones at those levels, but over

needs of "minority" users. Survey information may be used to reach some inferences about the distribution of such OS/FS community attitude regarding different categories of software, which in turn could be introduced as a dimension of inter-project diversity.

time, with the accumulation of a track record of successful submissions, would tend to migrate to lower branches of new trees.[42]

All of the foregoing complicating features of the resource allocation within and among OS/FS development projects are more or less interdependent, and this list is not exhaustive. There is therefore a great deal of challenging model-building work still to be done still, and still further empirical research must be devoted to obtain sensible parameterizations of the simulation structure. But we maintain that this effort is worth undertaking because we are convinced that OS/FS research activity, be it in computer science, economics or other social sciences, is now proliferating rapidly in empirical and theoretical directions, and some integrative tools are needed to better assess the findings and their implications. Empirical research of several kinds, about the nature of the involvement of developers in projects and their motivations, about the ecology of OS/FS projects as typically observed in SourceForge-like environments, about the commercial ecology and economy which now accompanies all successful OS/FS projects, not only should only be confronted with the model and its findings, but it should also orient further modelling advances.

As it is essential for theorists to engage in an continuing dialog with empirical researchers, agent-based simulation modelling would appear to provide at least part of the necessary language for conducting such exchanges. It is therefore to be hoped that by exploring this approach it will prove possible eventually to bring social science research on the free and open source model of software development to bear in a reliably informative way upon issues of public and private policy for a sector of the global economy that manifestly is rapidly growing in importance.

[42] The complex interplay of factors of learning and trust, and the ways that they may shape path dependent career trajectories of members of the OS/FS developer communities, have be carefully discussed in recent work by Mateos-Garcia and Steinmueller (2003).

References

Benkler, Yoachi, 2002. "Coase's Penguin, or, Linux and The Nature of the Firm," Version 04.3 (August), forthcoming in *Yale Law Journal*, 112 (Winter 2002-2003).

Bessen, James (2001) "Open Source Software: Free Provision of Complex Public Goods." Research on Innovation. Paper. [Available at: <www.researchoninnovation.org/opensrc.pdf>.]

Carayol, Nicolas and Jean-Michel Dalle, 2000. "Science wells: Modelling the 'problem of problem choice' within scientific communities", 5th WEHIA Conference, Marseille, June.

Dalle, Jean-Michel and Nicolas Jullien, 2000. "NT vs. Linux, or some explorations into the economics of free software," In: *Application of simulation to social sciences*, G. Ballot and G. Weisbuch, eds. Paris, France: Hermès, pp. 399-416.

Dalle, Jean-Michel and Nicolas Jullien, 2003. " 'Libre' software : turning fads into institutions?", *Research Policy*, vol. 32, No. 1, pp. 1-11.

Dasgupta, Partha and Paul A. David, "Information Disclosure and the Economics of Science and Technology" , Ch. 16 in *Arrow and the Ascent of Modern Economic Theory*, (G. Feiwel, ed.), New York: New York University Press, 1987, pp. 519-542.

Dasgupta, Partha and Paul A. David, 1994. "Toward a new economics of science", *Research Policy*, vol. 23, no. 5, pp. 487-521.

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

David, Paul A., 1998a. "Reputation and Agency in the Historical Emergence of the Institutions of 'Open Science'," *Center for Economic Policy Research, Publication No. 261*, Stanford University, (revised March 1994), further revised :December.

David, Paul A., 1998b. "Common Agency Contracting and the Emergence of 'Open Science' Institutions,"*American Economic Review*, 88(2), May.

David, Paul A., 1998c. "Communication Norms and the Collective Cognitive Performance of 'Invisible Colleges'," in *Creation and the Transfer of Knowledge: Institutions and Incentives*, G.Barba Navaretti et.al., eds., Berlin, Heidelberg, New York: Springer-Verlag.

David, Paul A., 2001a. "Path dependence, its critics and the quest for 'historical economics'," in *Evolution and Path Dependence in Economic Ideas: Past and Present*, eds. P. Garrouste and S. Ioannidies. Cheltenham, Glos.: Edward Elgar, 2001.

David, Paul A., 2001b. "The Political Economy of Public Science," in *The Regulation of Science and Technology*, Helen Lawton Smith, ed., London: Palgrave.

David, Paul A., Seema Arora and W. Edward Steinmueller, 2001. “Economic Organization and Viability of Open Source Software: A Proposal to The National Science Foundation,” SIEPR, Stanford University, 22 January.

DiBona, Chris, Sam Ockman, and Mark Stone, 1999. “Introduction”, In: *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, eds. Sebastopol, Calif.: O'Reilly & Associates, pp. 1-17.

Franke, Nikolaus and Eric von Hippel, 2002. “Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software,” MIT Sloan School of Management Working Paper No. 4341-02, January.

Ghosh, Rishab Aiyer, 1998. “Cooking pot markets: an economic model for the trade in free goods and services on the Internet,” *First Monday*, Vol. 3, No. 3 (March), [Available at : <firstmonday.org/issues/issue3_3/ghosh/index.html>.]

Ghosh, Rishab Aiyer, Rudiger Glott, Bernhard Kreiger and Gregario Robles, 2002. *The Free/Libre and Open Source Software Developers Survey and Study—FLOSS Final Report.* June. [Available at: http://www.infonomics.nl/FLOSS/report/].

Harhoff, Dietmar, Henkel, J. and von Hippel, E. (2000) “Profiting from Voluntary Information Spillovers: How Users Benefit by Freely Revealing their Innovations.” (July). [Available at: <opensource.mit.edu/papers/evhippel-voluntaryinfospillover.pdf>].

Kelty, Christopher M. (2001) “Free Software/Free Science.” *First Monday,* December. [Available at: <www.firstmonday.org/issues/issue6_12/kelty/index.html.]

Kogut, Bruce and Anca Metiu, 2001. “Open-Source software development and distributed innovation,” *Oxford Review of Economic Policy*, Vol. 17, No. 2, pp. 248-264.

Krishnamurthy, Sandeep. (2002) “Cave or Community? An Empirical Examination of 100 Mature Open Source Projects.” University of Washington, Bothell. [Available at: <faculty.washington.edu/sandeep>.]

Kuan, Jennifer, 2001. “Open Source Software as Consumer Integration into Production,” [Available at: <papers.ssrn.com/paper.taf?abstract_id=259648>.

Kuwabara, Ko, 2000. “Linux: A Bazaar at the Edge of Chaos,” *First Monday*, Vol. 5, No. 3 (March). [Available at: <firstmonday.org/issues/issue5_3/kuwabara/index.html>.]

Lakhani, Karim and Eric von Hippel, 2000. “How Open Source software works: ‘free’ user-to-user assistance,” MIT Sloan School of Management Working Paper No. 4117-00, May. Forthcoming in *Research Policy*. [Available at: <opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf>.]

Lerner, Josh and Tirole, Jean. (2000) “The Simple Economics of Open Source.” National Bureau of Economic Research (NBER) Working Paper 7600 (March). Available at: <www.nber.org/papers/w7600>.

Mateos-Garcia, Juan and W. Edward Steinmueller, (2003).”The Open Source Way of Working: A New Paradigm for the Division of Labour in Software Development?’, INK Open Source Research Working Paper No. 1, SPRU-University of Sussex, Brighton, England.

Merton, Robert K. (1973). *The Sociology of Science: Theoretical and Empirical Investigations.* Edited by Norman W. Storer. Chicago: University of Chicago Press.

Perens, Bruce, 1999. "The Open Source Definition," In: *Open Sources: Voices from the Open Source Revolution*. Sebastopol, Calif.: O'Reilly & Associates, pp. 171-188.

Raymond, Eric S., 1998b. "Homesteading the Noosphere," *First Monday*, volume 3, number 10 (October), [Available at <firstmonday.org/issues/issue3_10/raymond/index.html> and at <ww.tuxedo.org/~esr/writings/homesteading>.]

Raymond, Eric S., 1998a. "The Cathedral and the Bazaar," *First Monday*, volume 3, number 3 (March), [Available at: <firstmonday.org/issues/issue3_3/raymond/index.html>, and <ww.tuxedo.org/~esr/writings/cathedral-bazaar>.]

Raymond, Eric S., 1999a. “A Response to Nikolai Bezroukov, ” *First Monday*, volume 4, number 11 (November). [Available at: <firstmonday.org/issues/issue4_11/raymond/index.html>.

Raymond, Eric S., 1999b, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.*, Revised and Expanded Edition, Sebastopol, Calif.: O'Reilly & Associates.

Raymond, Eric S.,1999c. "The Magic Cauldron," in *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* Revised and Expanded Edition, Sebastopol, Calif.: O'Reilly & Associates.

Stallman, Richard, 1999. "The GNU Operating System and the Free Software Movement," In: *Open Sources: Voices from the Open Source Revolution,* C. DiBona, S. Ockman, and M. Stone, eds. Sebastopol, Calif.: O'Reilly & Associates, pp. 53-70.

von Hippel, Eric, 2001. “PERSPECTIVE: User toolkits for innovation,” *Journal of Product Innovation Management*, vol. 18, pp. 247-257.

von Hippel, Eric. (2002) “Horizontal Innovation Networks – By and For Users.” MIT Sloan School of Management (April). [Available at: <opensource.mit.edu/papers/vonhippel3.pdf>.

Weber, Steven. (2000) “The Political Economy of Open Source Software.” BRIE Working Paper 140, E-conomy Project Working Paper 15 (June). [Available at: <brie.berkeley.edu/~briewww/pubs/wp/wp140.pdf>.]