Motivation and Coordination in *Libre* Software Development
A Stygmergic Simulation Perspective on Large Community-Mode Projects

By

Jean-Michel Dalle
University Pierre et Marie Curie & IMRI-Dauphine
Jean-Michel.Dalle@upmc.fr

&

Paul A. David
Stanford University & Oxford Internet Institute
pad@stanford.edu

First draft: 1 December 2005
This version: 3 December 2007

ACKNOWLEDGEMENTS

The research reported in this paper was made possible by grant awards to Stanford University from the National Science-CISE DTS Program: IIS-0112962 (2001-04) and IIS-0329259 (2003-05). The international Project for Research on the Economics of Free/Libre & Open Source Software (*PREFLOSS*) has been sustained also by funding under these awards and local institutional support from the Stanford Institute for Economic Policy Research and its European academic partners: MERIT (at Maastricht University), IMRI (at Université Paris-Dauphine), Informatics-GSyC (at Universidad Rey Juan Carlos, Mostoles) and SPRU/INK (at University of Sussex). Our research has been supported also by *CALIBRE*, an EU FP6 Coordination Action. [See http://siepr.stanford.edu/programs/OpenSoftware_David/OS_Project_Funded_Announcmt.htm.]

In preparing this paper, we have benefited from discussions with participants at the EC "EPIP3"seminar convened at Scuola Superiore Sant'Anna in Pisa, Italy on 2-3 April 2004; and the OII-SIEPR "OWLS" workshop meeting organized by the Oxford Internet Institute in Oxford, England, on 25-26 June 2004. Many colleagues have had a helpful hand in shaping this paper: Rishab A. Ghosh, Jesus Gonzales-Baharona, Rudiger Glott, Israel Herraiz, Juan Mateos-Garcia, Gregorio Robles, Edward Steinmuller, and Andrew Waterman. In addition, we are grateful for comments and suggestions from the following participants at the EPIP3 and OWLS: Fabio Arcangeli, Matthijs den Besten, Robin Cowan, Brian Fitzgerald, Alfonso Gambardella, Bronwyn H. Hall, Jim Herbsleb, Eric von Hippel, Brian Kahin, Mathieu Lacage, Karim Lakhani, Jean-Charles Pomerol, Francesco Rullani, Walt Scacchi, and Joachim Voth. Georg Krogh and Eric von Hippel also read a previous draft and their criticisms and suggestions have contributed substantially to improving the present exposition. Such defects as have survived all this help are wholly ours.

ABSTRACT

This paper presents a stochastic simulation modeling tool used to study the implications of the mechanisms by which individual software developers' efforts are allocated within large and complex open source software projects. It thus illuminates the role of different forms of "motivations-at-the-margin" in the micro-level resource allocation process of distributed and decentralized multi-agent engineering undertakings of this kind. The modeling approach is informed by micro-level empirical evidence regarding the distribution of motivational profiles of developers that participate in large, community mode *libre* (free and open source) software projects, and the model is parameterized by isolating the parameter ranges in which it generates structures of code that share certain macro-level empirical regularities that are found to characterize the dynamics of actual projects. It is found that in this empirically relevant range parameter a variety of different motivations are necessarily represented within the community of developers. There is, further, a correspondence between the indicated mixture of motivations and the distribution of avowed motivations for engaging in FLOSS development, found in the survey responses of developers who were participants in large projects. Some implications for the "management" of large volunteer peer-production organizations are discussed in the conclusion.

Keywords: free and open source software (FLOSS), libre software engineering, maintainability, reliability, functional diversity, modularity, developers' motivations, user innovation, peer-esteem, reputational reward systems, agent-based modeling, stochastic simulation, stygmergy, morphogenesis.

1 *Introduction and Overview*

The research questions upon which this paper focuses concern the mechanisms that serve to direct and coordinate the work contributed by individual members of the distributed communities of developers associated with large, complex "free (in the sense of *libre*) and open source" software (FLOSS) projects.[1] This represents only one among the distinctive aspects of "the open source phenomenon," many of which have attracted the attention of software engineers, social and management scientists and legal scholars.[2] But, the organization of the voluntary work of many individuals in order to collectively produce large and technically sophisticated artefacts is particularly intriguing when considered from the economic resource allocation standpoint: it appears to proceed without the guidance of either the "visible hand" of a hierarchical management structure, or the "invisible hand" of a decentralized market-like price system.

Consequently, to understand the sources of both the capabilities and limitations of the mode of production exemplified by the array of visibly successful FLOSS projects such as Linux, Apache, Mozilla, Perl, MySQL, GNOME, and Open Office is not simply a matter of interest for those concerned with the relationship between this emerging segment of the software industry and the still dominant commercial producers of proprietary software packages. The "open source way of working" may turn out to have much broader application in the production of certain classes of information-goods. On that account, too, it deserves more systematic and sustained investigation than it has thus far received in the social and management science literature.

The motivations of those volunteering their time to produce and release open source software, and the means by which large number of such contributions are coordinated to produce coherent and reliable software systems have been topics of lively interest, and some perplexity. Indeed, much of the research initially devoted to the FLOSS phenomena by economists, sociologists and other social cientists, as well software engineers and students of organizational and management science, focused on this pair of questions. Although usually treated separately, the two actually are closely related, and the nature of their interconnections will be seen to have significant implications affecting the resource allocation mechanisms with this paper is concerned. They also have direct consequences for the macro-level properties of the resulting software systems, including the dynamics of code evolution, code maintainability and reliability – and possibly still other properties that affect the benefits afforded to end-users of the product, whether or not they have participated in its development.

The nexus between the motivation of individuals and their coordination in collective activities can be usefully studied within the framework provided by an agent-based simulation model, such as the one presented here. *SimCode* is a stochastic simulation tool that has been developed to illuminate and evaluate the social mechanisms by which individual software developers' efforts come to be allocated among the variety of software engineering tasks involved in large and complex FLOSS projects. The latter involve software development conceived from the outset as a "community" ("*C-mode*") production activity, and therefore differs importantly from the *independent* ("*I-mode"*) production of code by one or two individuals who release the program under open source licenses (see Dalle and David 2003, 2005).

Our model situates developers' choices about what contributions they will make within a

[1] We refer to free, libre and open source software interchangeably: for simplicity, we use FLOSS and "open source" both being now common in academic contexts. But in all instances we have in mind, unless otherwise indicated, software that is distributed under licenses conforming to the OSI definition, of which the GNU/General Public License is the form most widely used.

[2] See, e.g. Benkler (2002), Gonzalez-Barahona and Robles (2003), Koch and Schneider (2000), McGowan (2005), Scaachi (2003), von Hippel (2002,2004, 2005), Weber (2004).

technical environment that is created and continues to evolve from the interactions among successive individual choice-actions of that kind. We suppose that when a developer considers contributing to a given project at a given moment in time, he or she faces a structured set of software modules (or "code packages") whose respective technical properties and state of development (at that moment) are fully known. Individual selection decisions are assumed to be made in random order, each contributor contributing code to a particular "work package" of the project up to the limit set by their individual current code-writing capabilities ("effort endowments," specified in equivalent physical source lines of code). Subject to that constraint, each developer's selection is made where he or she perceives that effort is likely to yield the greatest "reward" -- directly or indirectly providing the greatest intrinsic or extrinsic source of satisfaction. The framework posits that a major systematic determinant of micro-level actions affecting and hence coordinating resource allocation within the project are the responses that are elicited, in effect, by the array of reward-promising signals that the agents receive at each point in time from state of the code itself.

Quite obviously, this modeling approach abstracts from the coordinating functions that may be performed by the flow of direct communications among project participants and potential contributors. The email lists and other associated electronic *fora* through which those communications latter take place have begun to attract systematic study, but to date only limited progress has been made in relating the apparent network patterns in those interactions to the concurrent or subsequent code-writing activities of the participants, and tracing their temporal co-evolution through the life of the project.[3] Until it becomes possible to analyze the email threads and discern what immediate or intermediating roles those communications may play in directing, or guiding participating developers' programming and patching contributions, we think it is reasonable to consider the impersonal coordinating signals that may be provided by the state of the code itself.

Economists may be inclined to view this abstraction as resembling the conventional conceptualization of a decentralized resource allocation mechanism in which atomistic agents respond to disembodied price signals generated by competitive markets, rather than interacting socially with one another. That particular analogy should not be pressed too strongly, however, because in the model we present there is nothing that corresponds to market prices, just as in the actual system whose properties we are trying to simulate there are no prices. A different and more informative analogy may be drawn with the mechanisms found in certain zoological systems where coordination of the actors' is achieved indirectly, in effect through signals imparted by the state of the artefacts that they collectively are fashioning. Such systems are referred to as "stygmergic," and recently have begun to be more widely studied, and applied in computational systems as forms of "swarm intelligence".[4]

The organization of the paper following this introductory overview is straightforward. Section 2 poses the questions about the open source mode of software production in their conceptual and analytical relationship to the preoccupation of much of the literature, which has been to try to identify one or another supposedly typical, or at least dominant motive that would explain the willingness to the mass of open source developers to volunteer their time skill in this pursuit Section 3 frames the present modeling approach empirically at the micro-level, by considering what has been revealed by surveys regarding the variety of reasons that FLOSS developers offer when asked to rationalize both their involvement in the activity and choices of specific projects in which to participate. The problem is reformulated, by pointing to the roles of motivations-at-the-margin rather than to fundamental human motives for social actions such as ego-satisfaction or altruism, materialism or ideology.

[3] See, e.g., von Krogh, Spaeth and Lakhani (2003); Crowston and Howison (2003, 2005). Robles and Gonzalez-Barahona (2005) present techniques that could be used integrate email list participation data with individuals' contributions to the project's code.

[4] See e.g., Bonabeau, Dorigo, and Theraulaz (2000); Breukner and Paranuk (2002).

"Motivations-at-the-margin" refers to the discriminating goals and preference-structures that affect individuals' local choices of what, where and how much effort an individual agent will to contribute, and hence the distribution of focused "micro-motives" that critically affects the workings of within-project resource allocation mechanisms. This context makes it particular important to bring what can be learned from the surveys of FLOSS developers to bear specifically upon the question of the motives animating those involved with large projects. The discussion therefore summarizes the findings of several independent empirical studies of developer motivations that have used cluster analysis methods to delineate principal configurations of motives among the respondents to the. An analysis (David and Shapiro, 2008) of the cluster assignments of individual respondents to the FLOSS-US (2003)[5] survey5 reveals that the mix of motivational clusters present among contributors to large projects differs significantly in several respects from the mixture observed among the sub-sample of respondents engaged on very small (*I-mode*) projects; furthermore, none of the identified clusters was overwhelmingly preponderant among the sub-sample of respondents that were active contributors to large FLOSS projects. This evidence lends support to the idea that rather than conceptualizing the participants as homogenous in their motivations, "project development communities" may be formed by individuals with a variety of quite different "micromotivations." It thus poses the question of how the assortment of motivations among the developer community attracted by a project shape the collective performance of the ensemble, and perhaps the projects ultimate success or failure. This question is tackled by means of simulation modeling in the remaining sections of the paper.

Our approach in constructing a parsimonious dynamical agent-based model is described in Section 4, which gives a mathematical representation of a particular deterministic discrete choice formulation of such a structure. The model allows for parametric variations in the weights assigned along two dimensions regarding the responsiveness of agents to "rewards" of different kinds, which are specific to contributions made to different sub-projects within the evolving code. The differential responsiveness on the part of agents can be interpreted as corresponding to differences in their motivational profiles. One parameterized response dimension distinguishes rewards based solely upon peer-regard for the degree technical merits inherent in contributions ("commits") to the project's code; whereas at the opposite extreme, rewards in the form of instrumental utility provided to individual developers by their respective contributions. The other dimension distinguishes been rewards that are individualistic, either in providing ego-gratification or utilitarian value for independent actions, contrasted with contributions whose nature requires more intense social interactions – and hence attraction to portions of the code (sub-projects) on which many others are engaged.

The four pure "pure types" of developer-motives that out schema can depict by extreme combinations of this pair of parameters (i.e., by values corresponding to the extrema of the two dimensional surface which the parameter-pair defines) are given the following colorful labels (a) "Raymondians"and Lerner-Tiroleans"—who are kudos-seeking programmers, capable of and attracted to working on the technically more demanding sub-projects for career reasons, not by community interactions; (b) "social hackers", who are drawn by technically challenging tasks that will bring community recognition and esteem (c) "social learners"– novitiates to FLOSS, who generally are less skilled programmers seeking community interactions for self improvement, and possibly self-actualization; (d) "von Hippelites" – who are individualist "user-innovators', not particularly drawn by intrinsic programming challenges in the projects infrastructure, but, instead, idiosyncratically attracted opportunities of building some particular functional capability that they will find more immediately useful.

Section 5 then presents the results of experiments using a particular stochastic specification of this simulation structure, in order to establish the effects on a project's development of alternative settings for that key parameter couplet. It is shown that they profoundly influence the probabilistic allocation of individual developers' contributions, thereby affecting the morphology of the evolving

[5] See http://www.stanford.edu/group/floss-us/index.html, and David, Waterman and Arora (2003).

code-tree. In particular, the joint specification of these two parameters -- which is to say, the relative strengths assigned to the four "pure motivational type" previously described – is found to control the model's ability to simulate the emergence of certain interesting macro-level features that appear (from recent empirical studies) to characterize structural and dynamical properties of the evolution of large open source software projects. Strikingly, in order for the model to generate an increasingly skewed distribution of module sizes and linearity in the time-trend of the project's overall file size, the necessary domain of parameter settings describe combinations of all four of the "pure"motivation-types.

The concluding section notices that the required intermediate domain in the parameter plane may be interpreted as corresponding to the mixture of the motivation clusters identified among survey respondents who had associated themselves with large projects. Hence the paper's title: "It takes all kinds." This is hardly advanced as a proof of necessity, let alone sufficiency for project growth, but rather as a first stage in the construction of a consistent vision of the generic features of "the open source way of working,"and one that raises a number of interesting questions about the possibilities of "managing" the allocation of volunteer peerproduction of complex information goods such as large software programs. But these findings are suggestive, at best. To underscore the qualifications that are warranted by the highly stylized nature of the simulation model in its present state of development, we close by review several directions in which future work – both ours and that of others -- will be required to make the *SimCode* framework a fully effective tool for integrating and assessing the implications of the growing body of empirical evidence about "micro-motives and macroperformance" of FLOSS development communities.

2. *What's so interesting about open source production?*

The initial contributions to the social science literature addressing the phenomenon of open source software 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.[6] 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 quest that has occupied observers, and especially economists, is to uncover the explanation for the evident success of open-source products in market competition against proprietary software – significantly on the basis not only of their lower cost, but their reputedly superior quality.[7] This problem 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.

However, it is difficult to ground the intense research interest that open-source software production currently attracts on a supposition of that the phenomenon is truly new and strange. Cooperative production of information and knowledge, among members of distributed epistemic communities who do not expect direct remuneration for their efforts hardly qualifies as new departure: there are numerous historical precursors and precedents for open-source software, perhaps most notably in the "invisible colleges" that appeared among the practitioners of the new experimental and mathematically approaches to scientific inquiry

[6] 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 (2002), Weber (2000), Kogut and Metiu (2001).

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

in western Europe in the course of the 17th century.[8] Thus, "open science" -- the mode of inquiry that emerged and became formally institutionalized during the era of the Scientific Revolution under systems of public and private patronage – provides an obvious cultural and organizational point of reference for observers of contemporary communities of programmers engaged in developing free software and open source software.[9] The "communal" ethos and norms of "the Republic of Science" emphasize the cooperative character of the larger purpose in which individual researchers are engaged, stressing that the accumulation of reliable knowledge is an essentially social process. The force of its universalistic 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.[10] 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 complementarities. This brings socially beneficial spillovers 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.

Open-source software production activities warrants systematic investigation, in our view, not just as a sub-species of the unusual class of technological objects called "computer software," but because the conjunction of a particular set of trends in the modern economy may give this development significant implications for the future of the advance of

[8] See, e.g., David (1998a, 1998b, 2001, 2004) and references to the history of science literature supplied therein. The professionalization of scientific research was a comparatively late development, as is well known; and as rapidly as that process proceeded since the late nineteenth century, the contributions of non-professionals in some domains of the exact sciences still have not been entirely eliminated. Optical astronomy provides a notable example in this regard: extensive communities of "amateur" star- and comet-watchers persist, and some among their members continue to score – and to verify – the occasional observational coup.

[9] This has not gone unrecognized by observers of the free and open software movements. In "The Magic Cauldron," Raymond (1999/2001) explicitly notices the connection between the information-sharing behavior of academic researchers and the practices of participants in open-source projects. Further, Raymond's (1999/2001) illuminating discussion ("in Homesteading the Noonesphere"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 rejoinder to N. Berzoukov's (1999) allegations on the point. Kelty (2001), and David, Arora and Steinmueller (2001), expand the comparison with the norms and institutions of open/academic science. Nevertheless, one should observe that that the parallel is by no means exact: formal professional accreditation and institutional affiliation is a salient *de facto* requirement for active participation in modern academic and public sector research communities, yet the computer programming and other software development tasks – whether in the commercial or the free and open source spheres -- remains an activity that has resisted becoming "professionalized."

[10] See Dasgupta and David (1994), David (1998b, 2002a, b) on the cognitive performance of open science networks in comparison with that of proprietary research organizations; David (2003) on the interaction between modern 'open science' and proprietary R&D.

knowledge, and consequently for knowledge-driven economic growth.[11] The first of those trends is that information-goods that share many of the special properties of software have been moving more and more to center-stage among the drivers of sustainable economic development. 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 advantages of an "open" (and cooperative) mode of organizing the generation of new knowledge – although long been recognized to have efficiency properties – has come to be increasingly appreciated, in contrast to the institutional solutions to the public goods problem that entail the restriction of access to information and research tools through secrecy or property rights enforcement. In contrast with the situation that prevails in most of the experimental and observational fields of research, where the conduct of open science encounters increasingly critical problems of providing public funding for professional researchers engaged in an increasingly costly pursuit, the mobilization of volunteered expertise from programmers who are able to communicate and collaborate essentially freely over an existing (Internet) infrastructure, makes the advantages of this way of organizing R&D activities all the more obtrusive.

Fourthly, and of great practical significance for those who seek to study it systematically, the open-source software mode of production itself is generating a wealth of quantitative information about this instantiation of "open epistemic communities." This latter development makes open-source software 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.

This re-framing of the phenomenon points toward a conceptual approach to the study of open source software production that highlights a broader, ultimately more policy-oriented set of issues than those which hitherto have dominated the literature. It is directed to answering an interrelated pair of canonical questions. First, we wish to know by what mechanisms open-source software projects mobilize the human resources, allocate the participants diverse expertise, coordinate the 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? Although these questions will be recognized immediately by management scientists and economists to be utterly familiar and straightforward, they have not yet having been explicitly posed or systematically pursued in the present context.

To pursue answers to these concrete questions compels a detailed examination of the actual workings of the system of social organization that actually allocates software development resources among the various software systems and applications projects that are being undertaken by "communities" of distributed and sometimes anonymous volunteers – as it is the situation of the large projects are found in the world of open-source software. How does the ensemble of developers collectively "select" among the observed array of projects that are launched? In contrast to efforts to illuminate the nature of the activities carried on by

[11] 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 open-source projects.

open source software development communities by reference to analogies with "bazaars," and "gift exchange societies," the tasks set by the foregoing approach represent an explicit response to the challenge of providing *non-metaphorical* answers to the classic questions of whether and how this instance of a decentralized decision resource allocation process could achieve coherent 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 and open source movements are enabling them to accomplish that outcome – without help either from the "invisible hand" of the market mechanism driven by price signals, or the "visible hands" of centralized managerial hierarchies.[12]

3. Characterizing open-source communities at work

SimCode can be described formally as a deterministic discrete choice framework, in which a sub-set of the parameters are interpreted as representing the force that the motivations of individual developers exert on the placement of their contributions among the "work-sites" within the project. The distribution of these underlying and probably heterogeneous motivations, and perceptions about the structure of differential rewards associated with various kinds of contributions, affects the way that the project's participants respond to the current state of the code. The pure forms of the motivations caricatured by those interpretations correspond to (1) the desire for the psychic or material reward derived from a "hacker" reputation based upon expert peer regard, and the ego-gratification of charismatic community leadership, or else "kudos"-seeking motivations (2) satisfactions to be derived from opportunities to that improve one's programming skills, and more generally social interaction-seeking motivations, or (3) enjoyment of the benefits of personally using the resulting software product, or else user-innovation motivations. Such reasons are salient among those given by survey respondents for their continued participation in open source projects, or are reported by expert participant observers of open source communities. Thus, the approach adopted here is to consider not whether such motives account for the amount of effort that is contributed voluntarily to the project, but, instead how such motives would bear upon the choices that developers make regarding the allocation of their incremental efforts, that is to say, to which among the existing modules (sub-projects) or new module the amount of code they can produce will be contributed: what we suggest to call "motivations at the margin".

Indeed, *SimCode* builds on the generic features of non-market social interaction mechanisms, which involve feedbacks 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. It has been structured with a view to its suitability for subsequent refinement and use in integrating and assessing the significance of empirical findings about patterns of resource allocation within large and more complex open-source projects, well known exemplars of which are the Linux operating system, the Mozilla web-browser and the Apache web-server. Systematic empirical evidence about the participants in such projects, their behaviors, patterns of communication and the internal modes of project organization has only lately begun to be collected.[13] Nonetheless, to guide

[12] 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.

[13] Pioneering studies of large projects include the work of S. Koch and G. Schneider (2000); Tuomi (2000, 2001); Dempsey et al. (1999, 2002); S. Krishnamurthy (2002). More recent studies have sought to exploit new methods of automated data-mining from source code repositories, and to build links between that information

the initial specifications it is possible to draw upon insights provided by experienced project leaders and descriptive generalizations about micro-level incentives from survey- and interview-based studies, regarding the nature of the community norms that might not only affect the mobilization of participants, but guide the allocation of software developers' efforts within particular projects.

(1) User-innovators

The first of the salient sets of "motivations at the margin" are the gamut of individual instrumental reasons that provide impetus to what may be described as "independent user-implemented innovations."[14] Indeed, this probably applies to the great mass of identifiably discrete open source software 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 outputs. The power of this motivating force obviously derives from the property of immediate efficacy, which has been noticed as a distinctive feature of computer programs. But, no less obviously, this class of motives 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, or those of a small number of collaborators. In our view, it is unlikely that a person writing an obscure driver for a either an obsolete or a newly-marketed printer that he wishes to use will be much concerned about the value that would be attached to this achievement by "the open-source software community." Individuals engaging in this sort of software development may regard themselves as belonging in every way to the free software and open source movements, and may be committed to using open source development tools.

A useful distinction here is made between FLOSS development in "community-mode", or for convenience *C-mode,* contrasting it with software production in purely "independent" *I-mode*. Inasmuch as *I-mode* products and producers, almost by definition, tend to remain restricted in their individual scope and do not provide as direct an experience of social participation, the empirical bases for generalizations about their code development procedures is too thin to provide interesting behavioral propositions that could be incorporated in a formal model distinct from that being developed here for the large, multi-agent collaborative

and data on communications flows among project participants. On patterns of authorship and the structure of code within large projects, obtained using the CODD data extraction algorithm (developed by R. A. Ghosh and V. V. Prakash (and described first to measuring the code size of projects in the Orbiten Free Survey (2000) [see http:www.orbiten.org/codd, see Ghosh (2003)]; for findings from the application of CODD to studies of sequential releases of the Linux kernel see Ghosh and David (2003). See Gonzalez-Barahona and Robles (2003, 2004); Robles, Koch and Gonzalez-Barahona (2004); Gonzalez-Baharona, Lopez and Robles (2004).

[14] 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. That "scratching a personal itch" is an important motivation among developers seems clear enough: of the 1562 respondents to the FLOSS-US Survey (see David, Waterman and Arora 2003), 56 percent scored as a "very important" or "important" the following reason for their participation: "I needed to perform tasks that could only be done with modified versions of existing software"; 53 percent assigned the same degrees of importance to: "I needed to fix bugs in software that I was using." But, whether individual user innovation typically gives rise to large community-based FLOSS development projects is another matter. The great mass of open source projects that appear on sites such as SourceForge and FreshMeat are small, and typically involve only a few identified developers at most. Indeed, considering the over 60 thousand project-groups listed on SourceForge in 2003, it turns out that 72 percent of them had only a single participant-member (private communication from Francesco Rullani, 13 December 2004). Whether the bulk of these small projects fit von Hippel's category of "user-innovators" cannot be determined from their size alone. Some might arise from code intended for commercial purposes that subsequently was released under open source license by the author.

projects.[15] Nonetheless, in the context of modelling resource allocation in *C-mode*, it is both appropriate and possible to make allowance for the possible role played by individual idiosyncratic use-motivations in affecting the randomly drawn developers' selections of the particular part of the large software project to which he or she is to contribute. As will be seen, this can be done quite readily, and without having to delineate the specific use-functions provided by the applications modules of the projects: we simply can add a stochastic component to the "peer regard" and the "learning community externalities" considerations that we model explicitly as affecting task selection behavior.

(2) Kudos-seeking and "Peer Regard"

Economists have suggested that career considerations, involving expectations of future material benefits from "signalling" expertise through open source contributions, are powerful in motivating the voluntary participation of developers (Lerner & Tirole, 2002). This plausible interpretative hypothesis rests largely on theoretical speculation, on analogies with the sociological studies of academic "open science" communities, and on preliminary empirical studies of open-source communities at work. Useful guidance in this respect can be drawn from another, less purely individualist framework for understanding recurring voluntary transactions: "The economy of regard" has been conceptualized by Avner Offer (1997) as a system of reciprocated exchanges that is situated "between the gift and the market." This is helpful in the present context for two principal reasons. It moves the discussion away from the essentially atomistic mode of analysis familiar in cartel theory and public finance economics, where agents are depicted as passively deciding whether to free-ride or join a pre-existing collective organization whose existence they view to be independent of their personal actions. Instead, it invites analysis of the intrinsic satisfactions that people may derive from interactive, community-creating aspects of participation – particularly in large open-source projects. Furthermore, the notion of the 'economy of regard' being distinct from the classical "gift economy," and positioned the intermediate space of non-market social systems based upon indirectly reciprocated, and partially personalized exchanges, may serve to characterize the larger open-source projects as belonging to a broader array of epistemic communities as institutionalized communities of practice.[16]

This framework accommodates the suggestion by Eric Raymond (1999) that the choices made by developers – particularly, choices regarding where and how to contribute their expertise and efforts – are guided by perceptions of the likely consequences in terms of recognition and esteem (including self-esteem) that those selections will elicit from those working on the projects in question, and from peers in the open source community at large. Raymond's argument, however, is considerably more restricted than the proposition that the voluntary participation of people in developing open source software is motivated by the promise of "reputation effects." Because it focuses on non-pecuniary, psychic satisfactions

[15] The present aspect of our investigations abstracts from the possible effects upon the organization of open-source production that may arise from choices among the licensing options. Such analytical attention as the question of open source software licensing has received from economists typically ignores the possibility of that consideration of such (production) effects might shape the licensing decision. See, e.g., Lerner and Tirole (2002), Gaudeul (2003). In a recent and interesting departure from the "mainstream" tradition, however, Gambardella and Hall (2004) examine a simple model that allows for the connection between the choice of licensing arrangements and the optimality of the production regime for information goods.

[16] One qualification might be noted, however: "regard" in Offer's discussion carries strongly personal and subjective connotations, whereas the term "esteem" is conveys greater objectivity and detachment, being used to signify appreciation for the accomplishments or professional qualities of another, rather than their person. See Brennan and Pettit (2004) on "the economy of esteem" in civil society and politics.

from the approbation of one's peers, it is not incompatible from the proposal to explain the phenomenon of voluntary participation in the collective production of software systems by suggesting that the people involved were "all" seeking material rewards that could be expected to flow from outside the open source community to those who had created a reputation within it, based upon their demonstrated expertise. However, focusing on regard is significantly more helpful in the context of a quest to understand in some detail how coordination and coherence is achieved in these distributed and largely unmanaged software projects. Basically, it suggests that there is a core of expert "hackers" who allocate their efforts in a quest for *kudos* that they believe will be forthcoming from their peers on the basis of the specific nature of the technical contributions that they make to the collective code-building enterprise. For this to serve as a mechanism of coordination and resource allocation, it would be important that the participants in the open source software community shared the normative valuation criteria for judging each other's technical contributions; that, being embedded in the same "economy of regard," they understood and accepted its specific norms, as well as being motivated by the same quest for *kudos*.

(3) Social interaction-seeking

It would be a mistake, however, to completely separate the issue of the sources of regard-oriented motivation with the 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. One must indeed go farther than this, however, by recognizing the potential importance of at least two other important classes of reasons why developers continue to participate in open source software projects. In their preliminary analysis of data from the FLOSS-US 2003 Survey, David, Waterman and Arora (2004) note the existence of a marked contrast between the tenor of responses to "general motivation" questions about developers' reasons for engaging in open source development work—questions of the same character as those posed by the FLOSS 2002 Survey (Ghosh et al. 2003), and a follow-on set of questions that asked why the respondent had chosen to work on their first project. The latter responses are significantly focused on instrumental considerations, most notably, the individual's interest in the technical characteristics of the project either for "skill development" or for personal use.

Further analysis of FLOSS-2002 Survey responses by Rudiger Glott (2004) found that among the majority of the respondents who gave initial reasons for participating in open-source software activities that were so diffuse as to resist categorization, the reasons they offered for continuing to participation display a bi-polar movement towards either more sharply delineated "ideological" commitment to the community, or a dominantly "instrumental" rationale. In the latter category, significantly, 'improving software skills" – already present among the focused motives for beginning participation -- emerged saliently among the reasons for sustained participation. Thus, we should recognize that neophyte developers may be strongly drawn toward working on sub-projects where the opportunities for learning from others, and beginning to contribute "patches" or simpler applications modules are most accessible, rather than absorbing and being guided by the peer regard criteria reflecting the open source "hacker ethos" described by Eric Raymond. This, one might conjecture, would induce them to gravitate toward sites in the evolving code where there is more activity, and a larger number of learner-peers among whom discussion of programming approaches and craft, as well as more abundant opportunities to observe ongoing code improvement. In a sense such clusters of activity provide the attractions of "learning community externalities."

Learning externalities would be likely to attract developers who are concerned primarily to improve programming skills and therefore will be motivated to contribute to those modules where there have been many previous contributions, a larger body of code to examine, and (most likely) a bigger sub-community of developers who are familiar with, and able to impart instructive information about technical aspects of the creation of the code present in that module. The argument for this interpretation carries even greater force in the case of neophyte developers, because access to a socially interactive and supportive learning environment are likely to be especially important at that stage, perhaps more than the particular technical features of the sub-project. Eventually, acquisition of software skills and corresponding improvements also should have a positive impact on future opportunities for gainful employment in the industry.

The foregoing discussion has drawn upon a variety (one might say a melange) of survey-based evidence and anecdotal observations about avowed and inferred motives of individual engaged in FLOSS development. While it may be adequate to portray the relevant range of activities that are in one way or another rewarding, it is uncomfortable not to have a more concrete view of the relevant frequency and intensity with which the various motivations (reasons for behavior) are expressed by those who actually are working on the larger FLOSS projects whose modes of internal coordination we seek to understand. Such a view is not offered by the entire distribution of responses to motivation-related questions in the large surveys discussed here, for the simple reason that the respondents include a very large proportion of individuals who appear to be engaged on very small projects, involving only one or two contributors.[17] Table 1, however, presents the finding of a study that unambiguously identified 176 individuals among the respondents to the FLOSS-US Survey as currently working on a project that had 30 or more members, and compared their motivational profiles with 214 other respondents whose current project had 1 or 2 members.[18] All the survey respondents' answers to the battery of questions about their reasons for participating in FLOSS activities were subjected to (agglomerative hierarchical) cluster analysis, which yielded the five distinctive clusters that are indicated in Table 1. The descriptive characterizations of clusters 1-5 are based on an analysis of the distribution of responses from the entire survey sample to a question that asked them to indicate how important each of eleven reasons for FLOSS participation was in their own experience.[19] Reasons that were accorded especially marked importance or unimportance (among the full sample of

[17] In the case of the FLOSS-US (2003) Survey, this may be inferred from the fact that of 1473 respondents who list a current project, 64.8 percent describe it as "unknown" or "slightly known" and 33.0 percent said they launched the project alone, while 46.8 percent reported that they launched it with others. Similarly, to 1055 respondents reporting the proportion of code they had contributed to their current project, 31 percent put that figure at or above 95 percent. (See David 2006 for further discussion.) Further confirmation is provided by the great mass of very small projects that appear on open source platforms such as SourceForge and FreshMeat: considering the over 60 thousand project-groups listed on SourceForge in 2003, it turns out that 72 percent of them had only a single participant-member (private communication from Francesco Rullani, 13 December 2004). Whether the bulk of these small projects are truly solo development efforts is not certain, as some may have originated in code previously written for commercial purposes and by subsequently released under open source license by the copyright holders.

[18] David, Shapiro and Waterman (2006) make use of information about current project names to identify the projects and link them to other sources, such the contemporaneous archives of SourceForge, from which the number of project members can be determined.

[19] The analysis constructed relative intensity measures of "importance" and "unimportance" on the basis of the distribution of answers to each of the 11 suggested motives, forming the ratio between the numbers within each cluster that answered "very important" and "important" and the number that answered "unimportant." The ratios for the several cluster was normalized for each suggested motive – dividing it by the corresponding ratio in the whole population. See David, Shapiro and Waterman (2006) for further details.

respondents assigned to each of the clusters) are indicated as the *distinctive* motives characteristics of the respective clusters.

(Insert Table 1 About Here)

What the Table presents are the distributions among the motivation-clusters of the sub-group of respondents that are known to have been working on the large projects. For purposes of comparison, a corresponding distribution is shown for the sub-group of respondents that were involved in the very small projects. Two points about these results deserve particular notice. First, looking at the distribution for the "community mode" developers, it is seen that those in clusters 2, 4 and 5 – each of which are marked by the technical interests and indications of expertise among their membership – are about equal in numerical weight.[20] Their combined weight exceeds that of the "social learners" in cluster 3, but not by very much. This is consistent with other less systematic reports that the large projects do indeed attract a significant proportion of comparatively less adept programmers, who are unlikely to be allowed much free rein to commit code in the more critical modules of the project infrastructure; they gain such experience as they may acquire among the mass of contributors seeking to commit code for driver subsystems and other similarly non-critical modules.[21]

The second point to note is that the large and small project participants do differ significantly in their distributions among the clusters.[22] "Aspiring hackers" and the more experienced "Raymondians" who like the challenge of fixing bugs and modifying the code they use, are relatively numerous in the large projects; whereas our "von Hippelites," who tend to have launched their own projects and are not motivated to learn about and patch other peoples' code, figure relatively prominently among the sub-population who are occupied with one- and two-person projects. But, it striking that that social interactions with like-minded individuals and the software skills-improving motives of cluster 3 types carry just about the same relative weight among the small project sub-sample as among the large. Obviously, neither a sense of participation in and self-identification with the open source movement, nor the opportunity to observe and learn from the source code of others' projects, requires being "embedded" in community-based FLOSS development.

4. SimCode – A Simulation Framework

The empirical insights gleaned in the preceding section furnish support for the idea that rather than conceptualizing the participants as homogenous in their motivations, "project development communities" may be formed by individuals with a variety of quite different

[20] Cluster 1 is a small and rather anomalous group which we have labelled "professionals". Apparently already expert (not being interested in improving software skills, or learning about new programs) they representing a bit under 5 percent of the total sub-ground on the large projects, and reject the importance of any of the intrinsic rewards and ideological reasons for collaborating in these projects. They assert, like many others, that it is important to them to be able to modify software code.

[21] On the control of commits excercised by maintainers in large projects that provide learning opportunities for the less expert programmers, see Kroah-Hartman's (2005) informative description of this important aspect of governance of the Linux kernel development process.

[22] David, Shapiro and Waterman (2006) report a value of 18.92 for Pearson's of Chi-square (df=4), which rejects the null hypothesis of no significant difference between the *I-mode* and the *C-mode* distributions in Table 1, at the .001 level.

"micro-motivations." This is an important point of departure for our modelling exercise, because the actions of individual developers associated with a large project lie at the core of the stochastic simulation structure that is employed to represent the way in which those actions are coordinated.[23] Our framework situates developers' choices as to what contributions they will make within a technical context that is formed and evolves from the interactions among successive choices of that kind. Thus, when a developer considers contributing to a given project at a given moment in time, we suppose that he or she faces a structured set of software modules[24] whose respective technical properties and state of development (at that moment) are fully known.[25]

Individual selection decisions of this kind are assumed to be made in random order, each contributor contributing code to a particular "work package" of the project up to the limit set by their current code-writing capabilities (their respective individual "effort endowments," which we take to be measured in equivalent physical source lines of code). Subject to that constraint, each developer's selection is made where he or she perceives that effort is likely to yield the greatest "reward" (directly or indirectly providing the most personal satisfaction or sense of accomplishment).

Our model thus posits that a major systematic determinant of micro-level resource allocation decisions – in the form of within-project task selection by developers – are the responses that are in effect elicited by signals that they receive at each point in time from state of the code itself. Quite obviously, this approach abstracts from the coordinating functions that may be performed by the flow of direct communications among project participants (and potential contributors) that take place via email lists and other associated electronic *fora*.[26] While economists may view our abstraction as resembling the conventional conceptualization of a decentralized resource allocation mechanism in which atomistic agents respond to disembodied price signals generated by competitive markets, rather than interacting socially with one another, the analogy should not be pressed too strongly. There is nothing in the present model that corresponds to market prices, just as in the actual system whose properties we are trying to simulate there are no prices, and another informative analogy may be drawn with the mechanism found in certain zoological systems that are referred to as "stygmergic."[27]

[23] The present version of this model has benefited from previous work on tree models (Carayol & Dalle, 2000) and from various criticisms and suggestions that were elicited from academic researchers and participant observers in open-source projects by our "early release" strategy of publication (Dalle & David, 2003). The goal, of course, remains not the achievement of "realism" but, rather the parsimonious representation of critical resource allocation mechanisms that, in this case, are confined to operating within a given large project.

[24] We use the term "module" or (code-)"package" interchangeably.

[25] This implies that each new contribution becomes immediately accessible to all developers, a simplification made in this version of the model. In reality there are variable lags between submissions and "commits" of code, and some submitssions of new code or of "patches" will not be accepted by the relevant project maintainer(s).

[26] To say that we have abstracted from such communications is not to say that we hold them not to be worth studying, or are prepared to rate these social interactions as unimportant. Quite the contrary for recent research integrating the study of email repositories and code repositories has yielded interesting and informative findings. See, e.g, Gonzalez-Barahona and Robles (2004).

[27] The term "stygmergy" was coined by the zoologist P.P. Grassé, as having been compounded from "styg" (for sign) and ergonomic (for work) in describing the complex interactions between the individual termites belonging to a colony that was constructing a mud nest, and the morphology of the materials in their constructed environment. More recently, the concept of stygmergy has come to be applied in the computer modelling of morphogenic processes in which the actors of a multi-agent system communicate indirectly through their environment (which is shaped by their collective actions), rather than via direct feedback among themselves: see Bonabeau et al. (1999); Therolaz and Bonabeau (1995). Derix, Simon and Coates (2003) survey of contemporary computer simulation models of this kind in the field of architecture. Robles, Merelo, and Gonzalez-Barahona,

3.1 Structure of the model

We give specific form to the argument advanced in previous sections about different motivations, for which we consider here the following further suppositions:

(1) Modules vary in their attractiveness to all developers in the population according to a commonly perceived reputational reward structure based on positive "peer regard." This follows the contention that there is such a shared normative structure related to the technical importance of distinct development tasks, which influences the individual developers' decisions about where in the project they will contribute.

(2) Developers have heterogeneous user preferences among all the existing modules (and potential modules that have yet to be created) for each module, which are stem from the anticipated "own-use" value of some particular application feature, or "affordance" of the project.

Combining (1) and (2) for developer *i* and module *m*, we have:

$$R^i{}_m = \rho_m + e^i{}_m \,,$$

where $R^i{}_m$ is the reward of contributing to module m for developer i, ρ_m is the regard-based reward and $e^i{}_m$ is the user-based reward.

We may assume, further, that the $e^i{}_m$ in the population follow the traditional assumptions of discrete choice theory (Anderson, de Palma & Thisse, 1992), and notably that this random variable is i.i.d.: hence, the probability of a given developer choosing module *m* can be stochastically described by:

$$\mathbf{P}\Big[\mathit{chosen\ module} = \mathit{module}\ m\,(m')\Big] = \frac{\rho_m(\alpha)}{\sum\limits_{m=1}^{\mathit{all\ modules}} \rho_m(\alpha) + \sum\limits_{m'=1}^{\mathit{all\ virtual\ modules}} \rho_{m'}(\alpha)}, \quad (0.1)$$

where the variable α is a measure of the incremental contribution, and we allow for the choice of initiating a new module by introducing the notion of "virtual modules", described more completely below. Hence, the more attractive modules will be selected, but only probabilistically because of the influence of heterogeneous intrinsic user-preferences among the existing (and virtual) modules.[28]

Open-source developers do not simply consider adding their efforts to existing modules, but may instead create new ones. To account for this important aspect of code growth, we invoke the following modelling finesse: suppose that to every existing module there is associated a 'virtual' module, which stands for the eventuality that a new module could be created from the existing one, either by developing an existing functionality out of it (in the form of an external module), or simply by adding a new one that would supplement the given module. Clearly then, the new module and the existing one would be technically linked.

(2005) present a FLOSS development simulation model on "stygmergic" principles, but one that is considerably closer to the ant- and wasp-behaviors, in that no connections are drawn with human motivation, differential responses to technical differences in code-writing projects, or project governance mechanisms of even the simplest sort.

[28] Further studies are needed, notably to introduce notably interdependent user preferences and more specific accounting of the attraction of applicative layers for users, turning the present structure into a full agent-based simulation framework.

Figure 1 represents the growth of a software system according to this rule: at each step, red lines and circles represented the last created module, while blues lines and circles represent virtual modules attached to each existing one, and black lines and circles represent older modules created during earlier steps.

(Insert Figure 1 About Here)

In this framework, the emerging architecture of the modules is mathematically a tree, since, by construction, there are no loops and each module is linked to only one (parent) module. This tree does not correspond exactly to either an actual directory tree, or to the full set of technical and functional dependencies known as the architecture of a software system (Bass, Clements & Kazman, 1998), since some of technical dependencies are obviously not accounted for. We would rather characterize it as an emerging architecture, which "stems" from the fact that developers generally decide to create a new module to solve a technical problem they face while working on a particular existing one, or as a development or part of an existing module. This emerging architecture here has much to do with what Herbert A. Simon famously characterized years ago in a seminal article on the "architecture of complexity" (1962), and this formulation is very much indebted to that conceptualization – all the more so as the emerging architecture that Simon considered is precisely a tree-like "hierarchical system."[29] Indeed, Simon suggested that the emerging architecture of complex systems tended to often be spontaneously tree-like, *because complex systems were born out of simple ones, and because simple systems then tend to be somehow included in more complex ones*. In our model of open-source software development, the rationale for the emergence of a hierarchy of modules is largely similar: a complex system is dynamically born out of a simple one; new modules are created out of existing ones to supplement them by developing existing functionalities or adding new ones; and these new modules can be included in higher ones during the compilation process or at least are called as sub-systems.[30]

3.2. Specification of the "peer regard" norms

Interpreting Raymond's (1999) and other observers' characterization of the normative structure of peer appraisal based on the technical nature of the contributions made by open source software developers, we posit that (statistically speaking) developers tend to prefer lower-level modules to higher-level ones in the hierarchical structure presented above, since the former, more general ones, are regarded as more generally relevant by their peers than more specialized ones, and also because their visibility being higher, it will automatically grant their contributors more regard from their peers. Contributing to the Linux kernel is deemed a potentially more rewarding activity than contributing to the file system, and the

[29] This should not be interpreted in the usual sense of hierarchy, which simply could describe an architecture comprising of several levels. Indeed, to avoid such mis-understandings, the French translation of Simon's paper has typically selected a word meaning "tree-like" (*arborescent*) to translate "hierarchy." A further connection between the tree-like architecture and Simon's work on semi-decomposable systems will be noted below in connection with the proposition that if multiple connections among the nodes at every level, and a fortiori from any level throughout the entire tree, additions of a module could impose costs that would inhibit that form of code expansion.

[30] We are also close here to the more recent research on modularity (Baldwin & Clark, 2000), and extending the model further in this direction, notably by studying more extensively, and modelling more accurately the actual technical interactions between modules, would also be a very fruitful research avenue.

latter still dominates writing an obscure driver for a newly-marketed printer. Stated differently, we postulate here that there is a strong dependency between the emerging hierarchical architecture of the software system and the associated hierarchy of peer regard. Yet in other words, we postulate that there is a lexicographic ordering of rewards based upon a discrete, mainly technically-based "tree-like" structure formed by the successive addition of modules. Clearly, this is an important assumption that should be tested empirically: in this respect, David, Dalle, Ghosh & Wolak (2004/2006) present preliminary elements in this direction, by showing that the pattern of signed and un-signed contributions in the Linux kernel is not random, and tends to show that technical dependencies tend to play a relatively significant role among other factors.

Still in line with the "peer regard" hypothesis, and to also account for the observations of Raymond and others, we add the two following "social norms" that are taken to affect developers' task-selection decisions:

[a] Launching a new project is more rewarding than contributing to an existing one, all the more so when several contributions have already been made: namely, the first contributions to a given module are more rewarding than later ones – this is more or less analogous to the "first to publish" rule in open science communities, and it seems to be also relevant in open source ones.

[b] Contributing to an active project is more rewarding that contributing to a stagnant or dormant one, as contributions will simply be more noticed by a larger number of peers, for it also is a relevant consideration for individuals seeking peer regard that one's contributions, however technically astute, should have an audience.

Translating this more into mathematical terms, we have:

$$\forall m \text{ a module: } \rho_m(\alpha) = r_m(x_m + \alpha) - r_m(x_m), \qquad (0.2)$$

where $\rho_m(\alpha)$ still stands for the expected[31] reward of contributing α to module m, $r_m(\bullet)$ is the cumulative reward function, i.e. the total reward associated with the sum of all contributions to module m, x_m is the current level of improvement of module m, i.e. precisely the sum of all past contributions, for instance measured in total number of added and deleted lines of code, and α is a potential contribution for a developer's given effort endowment expressed according to the same measure. Clearly then, by construction, for m' the virtual module associated with m:

$$\forall m': x_{m'} = 0 = r_{m'}(x_{m'}) \qquad (0.3)$$

Thus:

$$\forall m' \text{ the virtual module associated with module } m: \rho_{m'}(\alpha) = r_{m'}(\alpha), \qquad (0.4)$$

where $r_{m'}(\bullet)$ is a (positive) increasing concave function of the randomly drawn contribution, in conformity with rule [a] above, which imposes that the first contributions are more rewarded than the later ones.

We now specify, further, that:

[31] Part of the reward at least, especially for new modules, depends upon other contributions to be added later: therefore its expected nature.

$$r_m\left(x_m\right)=r_{d_m}\left(x_m\right)=v_{d_m}\left(x_m\right)d_m{}^{-\lambda}\left(\left(1+c_m\right)^{\gamma}\right), \tag{0.5}$$

where d_m is the distance of module m from the first "root" module; $v_{d_m}\left(x_m\right)$ is the function which gives the version number of module m at distance d_m from the root from its current improvement x_m; c_m is the number of contributions received by module m, and $\lambda \geq 0$ and $\gamma \geq 0$ are parameters.

This simplification of $r_m\left(\bullet\right)$ into $r_{d_m}\left(\bullet\right)$ is a direct consequence of the hierarchical and lexicographic assumptions in the reward system: the expected reward associated with contributing to module *m* depends on its location in the software architecture only to the degree of its vertical distance from the root of the code tree, d_m. This power of the attraction exerted by modules positioned closer to the role is governed by the magnitude of the characteristic exponent λ: when $\lambda = 0$ all modules are similarly rewarded, whatever their height d_m, whereas, as λ goes to infinity the rewards associated with contributing to modules closer to the root tend to be larger. Thus, we have:

$$r_0\left(x_{root}\right)=v_0\left(x_{root}\right)\left(\left(1+c_{root}\right)^{\gamma}\right) \text{ and } \forall m \neq root : r_m\left(x_m\right) \to 0 \text{ as } \lambda \to +\infty \tag{0.6}$$

Since the height of a virtual module is the height of its parent module plus one (by construction), equations (0.4) and (0.5) above imply:

$$\forall m : \rho_{m'}\left(\alpha\right)=r_{m'}\left(\alpha\right)=r_{d_m+1}\left(\alpha\right)=v_{d_m+1}\left(\alpha\right)\left(d_m+1\right)^{-\lambda}, \tag{0.7}$$

and

$$\forall m : \left(\left(1+c_{m'}\right)^{\gamma}\right)=1 \text{ since } \forall m : c_{m'}=0\,. \tag{0.8}$$

By construction, the term in c_m in equation (0.5) above allows us to account for rule [b], namely, to render the more active projects more rewarding for further contributions – even more and more so as γ increases, whereas this effect disappears completely when $\gamma = 0$. It is therefore not relevant for potential (virtual) modules, and the mathematical expression has been chosen in consequence of this.

We then define also:

$$v_{d_m}\left(x_m\right)=\log\left(1+x_m d_m{}^{\mu}\right), \tag{0.9}$$

where $\mu \geq 0$ is another characteristic exponent. Its sign implies that it is easier to increase version numbers for high modules than for lower ones. In all cases, one readily may verify that $v_{d_m}\left(x_m\right)$, and therefore $r_{m'}\left(x_m\right)$, are positive increasing concave functions of x_m, which is in line which our assumption that early contributions get more reward at every level in the tree.

To complete the description of the model it is necessary to specify the distribution of momentary "contribution capabilities," denoted by α. This can be viewed as arising from the convolution of two underlying distributions: the distribution of the time endowments of individual contributors, and the distribution of their individual code-writing efficiencies, measuring the latter by the amount of code (in physical lines of standard quality code produced per hour).[32] What the present version of the model specifies is the "reduced form" distribution that governs the size of the contribution submitted at each moment in the iteration. On the basis of the data gathered by the FLOSS-US Survey regarding the distribution of hours per week that respondents reported having working on their current open source project, it appears that a left-skewed specification of the "endowment distribution" is appropriate in this context.[33] For computational convenience, we posit that the instantaneous contributions (α) follow an exponential distribution, whence, by using the classical inverse transformation method on the cumulative distribution (e.g. Ross, 2003), we can compute the following exponential random number generator, which generates contributions α from a uniformly distributed probability:

$$\alpha = -\frac{1}{\delta}\ln\left(1-p\right), \qquad (0.10)$$

where, $p \in \left[0;1\right]$ is uniformly distributed and δ controls the mean of the distribution: a straightforward calculation will show that $\langle \alpha \rangle = \frac{1}{\delta}$, where $\langle . \rangle$ denotes the variable's mean.

Simulation experiments can then be run easily according to this model: at each discrete time step a new contribution is added to the existing system[34], i.e. either an existing module is improved or a new one is created. The simulation procedure is the following:

(a) A random contribution is given by eq. (0.10) ;

(b) The rewards of all existing modules, considering their current improvements, and of all virtual modules are computed according to eqs. (0.5), (0.7), and (0.9);

(c) A module is chosen according to eq. (0.1), and the system is then modified in consequence.

Figure 2 presents a typical result, in line with Figure 1, where each depicted tree is separated from its predecessor below by several time steps for the clarity of the exposition. Numbers are version numbers, which in the present version of the model do not account, although we have worked on more refined specifications of the model, the fact that in real software systems the change from version 1.9 to 2.0 is typically qualitatively different from the change from version 1.1 to 1.2, and more generally that software systems tend to conform to specific rules for numbering versions.

(Insert Figure 2 About Here)

[32] For the present version of the model no distinctions are made among contributions according to their purpose, i.e., whether they are "patches" submitted to correct bugs, or additions of new features – which Raymond (1998a) has characterized as "patches for bugs of omission". We do not account either for the involvement of commercial developers: we have started doing experiments in this respect, which will be reported elsewhere.

[33] See David, Waterman and Arora, 2003: p. 36 (Fig.48) on the distribution of hours per week, for which the mode (6 hrs.) and the median (7 hrs.) lie close together, and the mean of the upper half of the distribution is 15.8 hrs.; p. 38 (Fig. 49) shows that the distribution of maximum numbers of hours worked in a week on a single is far higher and more skewed: the mode being 31 hrs. and the median 39 hrs. On the supposition that individual productivity rates are correlated positively with time inputs (which cannot be said to be empirically established), the left-skew of the distribution of "endowments" –measured in equivalent lines of code—would be even more pronounced.

[34] As in all the experiments presented here, starting only with the root module with the initial improvement (number 1).

This framework can be used to analyse some of the properties of the emerging software systems, particularly the morphologies of emerging code trees and their sensitivity to parameter variations. In Section 5, we focus on a pair of parameters (λ, and γ) and interpret their varying configurations as corresponding to mixtures of the underlying developer population's "motivations-at-the-margin." That interpretation permits us to associate particular constellations of those motives with a corresponding set of morphological features of the code-tree – as generated by the corresponding probabilistic assignment function in eq. (0.1). And by the same token, it is possible to reverse the direction of inference: searching over the space of alternative combination of (λ, γ) permits calibration of the model so that it can simulate morphological features of the project's output consistent with "empirical regularities" that are observed in the evolution large open source projects. In thus delimiting an empirically relevant domain of the parameter space, we will have delimited the relevant domain of the underlying motives that should be viewed as serving to allocate the sequential contributions made by those participating in the project.

5. Simulating macro-level features of open source code evolution

By simulating the micro-level allocation dynamics it is possible to study their consequences for the emergent macro-properties the code that produced by a large open source project, considered in isolation. Therefore, it is pertinent to ask whether and what ranges of parameter values some macro-properties that have been observed to characterise actual projects of this kind can be reproduced by our simulation model. To the extent that there is a correspondence, we could reasonably view the latter as capable of being generated by the complex "stygmergic" mechanism that has been hypothesized, in which internal resource allocation is probabilistically guided by the evolving state of the code, because the latter elicits responses from developers who are "motivated-at-the-margin" by one or another of the several forms of expected "reward" that attach to contributing to various sites in the project.

We focus here on two such macro properties: linearity of growth and skewed-ness of the distribution of module sizes.[35] In exploring these questions, we focus attention on the influence of the two key parameters λ and γ, which respectively characterize the strength of the *kudos-seeking propensity*, and the *social interaction propensity* of developer community members. These interpretations can be elaborated with the aid of Figure 3.

In support this interpretation, we note that the effect of higher values of the parameter λ is to raise the relative reputational rewards, and hence the attractiveness of contributing to the lower-levels of the code tree. Thus, when considered in conjunction with the specifications of the norms on which "peer regard" is bestowed within the open source "hacker" community, λ in effect controls the strength of the influence on task selection of the exercised by the "*kudos*-seeking motivation". For present purposed, the latter embraces both the psychic, ego-gratification of Eric Raymond's (1999) hackers "homesteading the noosphere," and the material success-oriented participants who Lerner and Tirole (2003) depict as publicly signalling their expertise in order create an external reputation – reflecting the peer regard of the hacker community – in the expectation of subsequently garnering economic payoffs. As different in character as those motives may be, operationally both would have the effect of focusing the agents' contributions on modules that were closer to the

[35] In future works, we intend to move toward more detailed analyses of the properties of code architectures.

root of the code-tree; and neither type would be particularly attracted to "follow the crowd," i.e., to contribute to modules where much code already had been written. Accordingly, we have marked the upper left-hand corner of the parameter space in Figure 3 to be to home ground for contributions generated by "*Raymondian*" and "*Lerner-Tirolean*" micro-motivations.

The parameter γ, on the other hand, raises the relative attraction exerted by modules to which there have been many prior contributions, and where, because many individuals can be presumed to have been involved and acquired knowledge of that particular code structure and its development, there are greater apparent opportunities for social interactions and particularly for interactive learning and skills-improvement which (as was noted in Section 3) figures among the salient reasons that survey respondents cite for their continuing participation in open source development activities. Higher values of γ may be interpreted as associated with the attraction of sites where there is greater scope to for developers to enter readily and enjoy the benefits more frequent interactions with others who are closer to their own stage of expertise.

This points to the lower right-hand corner of the parameter plane in Figure 3 as the locus of the pure "social interaction and skill-seeking neophytes, because that is where the kudos-seeking attraction of working at the more technically important and/or complex modules closer to the root of the code-tree is weakest. It follows that upper right-hand corner of the λ - γ plane represents the locus of concentration of more expert seekers of *kudos* who derive physical rewards not only from peer approbation of their technical achievements, seek to establish a charismatic leadership role by working on sub-projects where others are engaged. For the sake of brevity we affix the label "social hacker" to this corner of Figure 3. It remains to note that where the motivation for peer regard is weak, and where individuals are induced to make contributions more by the prospects of producing specific software applications from whose use they anticipate enjoying a stream of benefits, we would expect to find the purest form of the "user-innovators" who figure prominently in the research of Eric von Hippel and his co-authors that addresses the phenomenon of open source software.[36] Obviously, then lower left-hand corner of the parameter space must the home base of "*von Hippelites*".[37]

(Insert Figure 3 About Here)

4.1 Simulation results on 'linearity' in code growth

The macro-level regularities observed in the evolution of computer software has formed a subject of interest among software engineers that extends back over some three decades to the pioneer studies by Lehman (1980, 2000), who adduced a number of empirical "laws" of software development. On the basis of observations from commercial development projects on the pace at which the number of files grew in the period following the initial release of product, Lehman's "Fourth Law" revised by Turski (1996), held that the pace at which files were added was close to linear but tended to slow in absolutely as well as in

[36] See Harhoff, Henkel and von Hippel (2000), Lakhani and von Hippel (2003).

[37] It should be stressed that this is the "pure form" of the species. In von Hippel (2004, 2005) the point is made that user-innovations often "freely reveal" details of their inventions (certainly so in the case of open source software), and this is explained by resort to a mixture of essentially instrumental utilitarian rationales, including reputation-building a la Lerner-Tirole (2003), and garnering interest and possibly external incremental applications of the new product.

proportion to the existing code base.[38] Among expectations that could be offered in support of this assertion, the argument could be that the number of possible interconnections among *n* files would increase approximately as the square of their number, and this would tend to absorb a rapidly rising amount of effort in writing and debugging, thereby slowing the pace at which further files could be added.[39] A further implication of this view was that those projects that were able to mobilize the additional resources to keep developing new code, would tend to produce an unmanageably complex and interconnected mass of files, and in the end would encounter overwhelming maintenance problems.

The issue of code evolution in large software projects recently has drawn renewed attention, stimulated by the emergence and continuing, readily visible growth of the code of widely known open source projects like Apache, Mozilla, GNOME, and Linux. There is now evidence that a super-linear growth profile characterizes the continuing evolution of some of the sub-directories of the Linux kernel.[40] It has also been shown recently that conformity with a linear pattern in the temporal evolution of file size is quite general among many less exceptional but nonetheless substantial "open-source" software systems that are being developed in *C-mode*. Linearity in this context means constancy of the number of new files added in each interval of time. In Robles et al. (2005) are presented the results of a recent statistical study of the evolution of file sizes for 18 open-source projects: 15 of them exhibit a linear regression fit with R^2 values above 0.90; in the case of 10 projects the linear fit is even closer, leaving only 5 percent of the variance unexplained. We therefore take "linearity" as a first target at which to aim in setting empirically relevant parameters for our simulation model.

Figure 4 displays the results of searching the (λ, γ)–space to discover whether linear growth profiles emerge with various settings of these parameters. For each setting of those parameters (holding the other parameter values unchanged), ten simulations were run and the time path of the average count of the number of nodes, or modules in the graph describing the code tree, was obtained. Figure 4 summarizes the results by plotting the value of the correlation coefficient R^2 obtained by fitting a linear regression model to the averaged time

[38] See the discussion of this literature in Scacchi (2003) and Robles et al. (2005); particularly the convex growth profiles – following an inverse square law – that are reported by the statistical analyses of proprietary commercial project by Turski (1996), and the recent reconsideration of the topic by Lehman (1997). The IMB 360/OS exhibited a growth profile that was linear in the number of releases, which was a notable exception among the small number of commercial software systems that were examined in the early studies.

[39] This argument differs from the class "span of attention" problem that is likely to rise if control of commitments is hierarchically controlled with final authority residing in a single "maintainer," a situation that appears to have been confronted, and overcome by the Linux kernel project, soon after the number of physical lines of code (in release 2.1) passed the 1 million mark. Larry McVoy, in an oft-cited email dated 30 September 1998 (see McVoy et al. 1998) wrote:

"The Problem ----

"The problem is that Linus doesn't scale. We can't expect to see the rate of change to the kernel, which gets more complex and larger daily, continue to increase and expect Linus to keep up. But we also don't want to have Linus lose control and final say over the kernel, he's demonstrated over and over that he is good at that.

"The basic solution ----

"Figure out a means by which Linus can surround himself with some number of people who do part of his job. Add tools which make that possible."

[40] See the extensive analysis by Godfrey and Tu (2000), which also surveys the background of previous findings regarding commercial software evolution, against which the case of Linux stands out as particularly anomalous, and also later results by Robles et al. (2005).

series for each of the sets of parameter-settings employed.[41] What it makes evident is that our simulation model embodies structural features which enable it to generate the characteristic linearity of code evolution of open-source software projects with a wide range of values of λ and γ. Although by itself this does not allow us to tightly fix the domain of "empirically relevant parameter values" it is in another respect very reassuring to find that linearity can be achieved with a wide range of values for the couplet (λ , γ): as we previously have pointed out, the latter can be interpreted as reflecting many different combinations of underlying motivation-types among the members of the developer community attached to the project.

(Insert Figure 4 About Here)

4.2 Simulation results on the distribution of module sizes

A way to further characterize the morphology of the code is to compute the Gini coefficient of the distributions of the sizes of modules – the tree's "leaves". The reason for this is an interesting empirical finding about the distribution of the sizes of the modules within large projects. Ghosh & David (2003), in their study of the development of the Linux kernel, reproduced here as Table 1, show that the distribution of module sizes is very skewed, with 10 percent of the modules accounting from more than half of the projects total code (measured in bytes). Moreover, the Gini coefficients of the distribution of module sizes tend are not only high from an early stage of the project's development (Release 1.0.3), but they continue to rise. Although these results were obtained for a rather exceptional project (however emblematic the Linux kernel may be), they appear to characterize other large open-source projects, such as GNOME. This relatively striking feature means that there is a limited number of modules with receive numerous contributions, and a large number of modules with only a limited number of contributions.

Figure 5 presents the Gini coefficients measuring the degree of concentration of the module-size distribution obtained from simulations which hold constant all the parameter except the pairs (λ,γ), varying the latter over the whole of the potentially relevant domain. Clearly, the region of the parameter-space within which these parameter-couplets (which we interpreted as reflecting the strength of "kudos-seeking" and "social interaction and skill-seeking" motives, respectively) powerfully raises the degree of concentration in the distribution of the project's code. There is a fairly narrow the region in which the Gini coefficients exceed 0.7, and this surrounds the "ridge-line" that appears in Figure 5.[42] Notice

[41] All the simulations reported in this section have been conducted with these parameter values :

$$\begin{cases} \delta = 3 \\ \mu = 0.5 \\ \theta = 0.5 \\ \xi = 2 \end{cases}$$

Similar results seem to hold generally for other values, and we focus here on showing the emergence of macro-properties as a robust feature of the basic structural specifications of the model, depending upon the foregoing plausible fixed values, and exploring the the effects of variations in λ , and γ .

[42] Along that line are found maximum concentration measures for ***G*** in the range 0.86-.88.

that the simulation experiments reveals the existence of a linear combination of λ and γ that defines a "project viability" limit; above this boundary, the software system fails to develop and leaves essentially all the code growth confined to a single (root) module.[43] Furthermore, these results tend to suggest that the evolution Linux, as reported in Table 1 above, would correspond to values of γ close to or greater than 1, i.e. to developers' task selections being guided by rather strong propensities for social interaction and skill-improvement.

(Insert Table 2 About Here)

To these results may be added several others, from simulation exercises that are not displayed here:

i. High Gini coefficients do not emerge when growing software systems without rule [b] above, i.e. if we set $\gamma = 0$, so that modules do not attract further contributions on the basis of the number of contributions they previously received (as indicated by their version number);

ii. High Gini coefficients do not emerge when rule [b] is replaced by an alternative specification that allow the accumulation of many offshoots from a given module to reduce that module's attractiveness as a site for further contributions;

iii. Gini coefficients tend to increase with time as the size of the code grows, but again conditional on the value of γ being sufficiently big, a finding that is not inconsistent with the preliminary evidence presented by Table 1 for the case of the Linux kernel.

4.3 The calibrated model and its implications about micro-motivations

The foregoing results from simulation of the macro-properties large open-source projects carry implications about the empirically relevant domain of the parameter space in our model. First we can project onto the (λ,γ)-plane of Figure 3 the empirically relevant broad parameter domain in which code evolution exhibits linear time trends for the count of modules in the project. The result appears in the transparent darker area on the left and upper side of the plane in Figure 6, bounded by an arc that descends from the top right corner. Next we project the domain of relevantly high Gini coefficients – corresponding roughly to the region of Gini-values upwards of 0.7 – onto the same plane. This forms the dotted-blue band, sloping downwards from left to right in the middle of the (λ,γ)-plane depicted in Figure 6. The zone in which the two projects intersect forms the tilted lozenge, marked with a wiggly green line in the figure. This overlay on the space indicating the micro-motive interpretations of the parameter values (displayed in Figure 3) points to a number of implications about the micro-motives of the contributors to our simulated open source software project.

The results reinforce the central message emerging from the (previously cited) survey inquiries into the motives that impel developers to contribute to a particular open source community's project. It not appropriate to think of the communities mobilized by the large projects as monolithic, or even as reflecting a single salient motive which can be said to typify their participants. The implication that *mixtures of motives* must characterize these

[43] A close approximation to this boundary line is found as: *max-Gini* = (0.1) λ + (0.43) γ .

communities is plausible and reassuring. There are good empirical grounds for thinking that "core developers" are at work on these projects, and it is reasonable to characterize them in the way that Raymond (1999) does as being concerned primarily with the esteem in which they are held by expert peers in the hacker community. Whether this matters for instrumental, career-related reasons, including the prospective material rewards that Lerner and Tirole (2002) see as inducing them to volunteer, or whether the spur is the personal ego-gratification of gaining "peer regard," is not really consequential in this connection. Similarly, one must acknowledge that large projects also mobilize a periphery of the "less expert" participants who are attracted to contribute to sub-projects that promise to enhance their skills in programming, or satisfy their desire for an involvement in social interactions directed to a practical collective purpose. The results of the calibrating the model point militate against accepting either or both of those "pure types" as characterising the motivational drivers of the members of FLOSS development communities. Correspondingly, there is rather more room in the picture for both "social hackers," on the one hand, and, on the other hand, individualists who are drawn to work on specific applications that would provide tools that meets their immediate and idiosyncratic needs.

5. Conclusions, Qualifications and the Way Forward

The intermediate positioning of the empirically relevant domain – within the plane of the parameter-pair identified in Figure 6 – may be interpreted legitimately as telling us that the populations of developers forming the "intelligent swarms" at work on code of the larger FLOSS project's comprise various "mixtures" of individuals actors who may be loosely categorized as having distinctive motivational profiles. It is therefore not surprising to notice that this view is entirely consistent with the impression obtained from Table 1, which showed the distribution of FLOSS-US survey respondents working in larger project communities among the motivational profiles extracted by cluster analysis. Although some of the labels we have affixed to those empirical clusters coincide with the "pure type" characterizations at the corners of the parameter-plane in Figures 3 and 6, it is not possible to draw any strict correspondence between the parameter domain and the empirical weights of the motivational clusters in Table 1, because in this reduced form models the individual agents are not distinguished from one another by their susceptibilities to "rewards" for contributions of different kinds, made at different places in the evolving code map of the project.

This paper has described the rationale and results of the most recently completed stage in the construction of a stochastic model that can simulate the process of open source software development in "community-mode". A quite lengthy list of questions of analytical interest – questions that clearly could be pursued by elaborating the model in various ways, already has been compiled in the previously published reports on this undertaking.[44] It appears now that there are three main directions in which the next phase of the project could most usefully proceed. One path would continue towards elaborating the present framework of the model to permit more precise calibration and simulation of a greater number of aspects of the evolving code of a single large project. This would increase the relevance of this modelling tool by taking into consideration a greater number of empirical regularities in regard to the types and sizes of modules, the structure of technical dependences among modules, the overall architectural morphology of the code, and so forth. An alterative branch on the same empirical path would take the model towards accommodating greater micro-level detail its representations of the behaviors of the individual project participants. By identifying the simulated agents, and tracking the history of their contributions, and perhaps the latter's

[44] See Dalle and David (2003/2005, 2006); Dalle, David, den Besten and Lacage (2006).

relationships to technical features of the code, it would be possible also to address other questions, such as the sources of the highly skewed distributions of contributions to the code of these community-based projects. There is yet another obvious and important path to follow in the next phase of model-building: taking a discrete step from simulating the generation of an isolated code-tree to the simulation of a forest of such trees.

Looking ahead on the several indicated trajectories for future elaboration of the model, it seems obvious that it will be beyond our power to adequately pursue on our own even the principal items in the vast research agenda that is opening before us. At least, it will not be possible to go forward at a pace that can keep up with the proliferating sources of empirical data that a fully specified model could illuminate, or with the multiplying policy questions that a carefully parameterized model could be used to analyze. Consequently, in a somewhat self-referential fashion, we are moving towards facilitating the conduct of the simulation project in the distributed open-source manner. The multi-tree release of the model will provide the source code we have used in exercising the model so that others may use to replicate our results and further modify and elaborate the structure. Whether this should formally become an experiment in organizing this kind of research on open-source as an open-source-like project – with all that this implies about claims to copyrights, licensing terms and governance norms – raises some interesting and arresting questions that at this point have not been resolved. Collaboration is far from costless, and the resources and ingenuity that FLOSS projects must devote to managing their growth are more likely to find the appreciative recognition and representation that they undoubtedly deserve within this modelling framework if the open-source and "open science" modes can indeed be fused in the reformulation of *SimCode* as a platform on which others can build.

REFERENCES

Anderson, de Palma and Jacques-François Thisse. 1992. "Discrete Choice Theory of Product Differentiation." Cambridge, Mass.: MIT Press.

Baldwin, Carliss Y. and Kim B. Clark. 2000. Design Rules: Vol.1. "The Power of Modularity." Cambridge, Mass: MIT Press.

Bass, Len J., P. Clements and Rick Kazman. 1998. "Software Architecture in Practice". Addison-Wesley.

Benkler, Yochai. 2002. "Coase's Penguin, or Linux and the Nature of the Firm." Yale Law Journal. 112: 369-446.

Bonabeau, E., M. Dorigo, and G.Theraulaz, 2000. "Inspiration for optimization from social insert behaviour," *Nature*, 406: pp. 39-42.

Boston Consulting Group. 2002. *Survey of free software/open source developers conducted by the Boston Consulting Group*. [Available at: <http://www.osdn.com/bcg >.]

Brennan, Geoffrey and Philip Pettit. 2004. *The Economy of Esteem: An Essay on Civil and Political Society*, Oxford: Oxford University Press.

Breukner, S. A. and H. van D. Paranuk, (2002), "Swarming agents for distributed pattern detection and classification," In Proc. Workshop on Ubiquitous Computing. First Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 02), Bologna, Italy. August.

Crowston, K. and J. Howison, 2003. "The social structure of open source software development teams." In *OASIS 2003 Workshop (IFIP 8.2 WG).* Available at: http://floss.syr.edu/.

Crowston, K. and J. Howison, 2005."Hierarchy and centralization in free and open source software team communications. Forthcoming in *Knowledge, Technology and Policy*, Special Edition on Free, Libre and Open Source Software Communities. Available at: http://floss.syr.edu/.

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

Dalle, Jean-Michel. 1997. "Heterogeneity vs. externalities: a tale of possible technological landscapes", *Journal of Evolutionary Economics* 7: 395-413.

Dalle, Jean-Michel and Paul A. David. 2001. "On open source software and the organization of cathedral-building: metaphors and realities." Working Paper, SIEPR-NOSTRA Project on the Economics of Open Source Software, December, revised version submitted to First Monday.

________.2003/2005. "The Allocation of Software Development Resources in 'Open Source' Production Mode," Ch. 14 in *Perspectives on Open Source Software*. eds., J. Feller, B. Fitzgerald, S. Hissam, K. Lakhani. Cambridge MA: MIT Press. Electronic pre-print (2003 SIEPR-Project NOSTRA Working Paper) available at: http://siepr.stanford.edu/programs/OpenSoftware_David/NSFOSF_Publications.html.

________.2006. "Simulating Code Growth in Libre (Open-Source) Mode," forthcoming in *The Economics of the Internet*, eds., Eric Brousseau and Nicola Curien. Cambridge: Cambridge University Press, (Fall) 2006.

Dalle, Jean-Michel, Paul A. David, Matthijs den Besten and Mathieu Lacage, 2006. "From micro-motives to macro-performance in open-source software projects: A social welfare assessment using *SimCode*.", presented at Complexity 2006 Conference, Aix en Provence, France, May 18-20th.

Dalle, Jean-Michel, Paul A. David and W.E. Steinmueller. 2002. "An Agenda for Integrated Research on the Economic Organization & Efficiency of Libre Software Production." Available at: http://siepr.stanford.edu/programs/OpenSoftware_David/FLOSS%20Conf%20Stmt_JMD+PD+ES_v6.htm.

Dalle, Jean.-Michel, P. A. David, Rishab A. Ghosh, and W. E. Steinmueller., 2005. "Advancing Economic Research on the Free and Open Source Software Mode of Production," In *How Open is the Future? Economic, Social & Cultural Scenarios Based On Open Standards*, Marleen Wynants and Jan Cornelis, Eds., Brussels: Vrjie Universiteit Brussels (VUB) Press) [Pre-print as SIEPR Discussion Paper (December 2004) available at: http://siepr.stanford.edu/programs/OpenSoftware_David/NSFOSF_Publications.html.]

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*, 32(1):1-11.

Dasgupta, Partha and Paul A. David. 1987. 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.

——— 1994. "Toward a new economics of science", *Research Policy*, vol. 23, no. 5, pp. 487-521.

David, Paul A. 1998a. Communication Norms and the Collective Cognitive Performance of 'Invisible Colleges in *Creation and Transfer of Knowledge: Institutions and Incentives,* Physica-Verlag Series *Contributions to Economics*, G.Barba. Navaretii et al., eds., Berlin, Heidelberg, New York: Springer-Verlag.

——— 1998b. Common Agency Contracting and the Emergence of 'Open Science' Institutions, *American Economic Review*, 88(2): 15-21 (May).

——— 2000. "Patronage, Reputation, and Common Agency Contracting in the Scientific Revolution: From Keeping 'Nature's Secrets' to the Institutionalization of 'Open Science." (Unpublished; under review at *Journal of Economic History*).

——— 2001. "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.

_____ 2002a. "La coopération, la créativité et al clôture des débats dans les sciences, in Institutions et innovation: De la recherche aux systèms sociaux d'innovation." (Sous la direction de Jean-Phillipe Touffut) Paris:Bibliothèque Albin Michel Economié, pp. 67-104.

_____ 2002b. "Cooperation, Creativity and Closure in Scientific Research Networks: Modeling the Simpler Dynamics of Invisible Colleges," SIEPR/CEEG-Social Science and Technology Seminar Series Paper (December 4, 2002). [Available at: http://siepr.stanford.edu/programs/SST_Seminars/David_All.pdf].

_____ 2003. "The Economic Logic of 'Open Science' and the Balance between Private Property Rights and the Public Domain in Scientific Data and Information: A Primer," in National Research Council, *The Role of the Public Domain in Scientific Data and Information*, Washington, D.C.: National Academy Press/

_____ 2004. "Understanding the Emergence of Open Science Institutions: Functionalist Economics in Historical Context," *Industrial and Corporate Change*, 13(1), August.

David, Paul A. 2006. "A multi-dimensional view of the sustainability of free and open source software development. Presented at the *OSSWatch Workshop on Open Source and Sustainability*, held in Oxford, 10-12 April. [Available at: http://www.oii.ox.ac.uk/research/files/OSSWatch_2006_Sustainability&FLOSS_txt4+slides0506.pdf.]

David, Paul A., Jean-Michel Dalle, Rishab Aiyer Ghosh and Frank Wolak. 2004/2006. "Free and Open Source Software Developers and 'the Economy of Esteem': A Quantitative Analysis of Code-Signing Patterns within the Linux Kernel," Revised version of the June 2004 paper, previously available at: http://siepr.stanford.edu/programs/OpenSoftware_David/Economy-of-Regard_8+_OWLS.pdf .

David, P. A., J. Shapiro and A. H. Waterman, 2006. "What is known about the motivations of the contributors to large FLOSS projects?," Stanford-SIEPR Open Source Project Working Paper, April.

David, Paul A., Andrew H. Waterman and Seema Arora .2003. "FLOSS-US: The Free/Libre Open Source Software Developer Survey for 2003: A First Report." (September) [Available at: http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf.]

Dempsey, B., Weiss, D., Jones, P. and Greenberg, J. 1999. "A quantitative profile of a community of open source Linux developers." SILS Tech. Rep. TR-1999-05, School of Information and Library Science, University of North Carolina at Chapel Hill, (October). [Available at: www.metalab.unc.edu/osrt/.

_____ 2002. "Who is an open source software developer?" *Communications of the ACM,* Volume 45, Number 2 (2002), Pages 67-72.

Elliott, Margaret. S. and Walt Scacchi. 2003. "Free Software Development: A Case Study of Software Development in a Virtual Organizational Culture." April. http://opensource.mit.edu/papers/eliottscacchi.pdf

Feller, Joe and Brian Fitzgerald. 2002. "Understanding Open Source Software Development." Addison-Wesley: UK.

Franke, Nikolaus and Eric von Hippel. 2003. “Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software.” *Research Policy*, 32 (7): 1199-1215, Special Issue on open source software development edited by Georg von Krogh and Eric von Hippel.

Gambardella, Alfonso and Bronwyn H. Hall. 2004.”Proprietary vs. Public Domain Licensing of Software and Research Products.” Working Paper. Scuola Superiore Sant’ Anna, Pisa. February. (Revised version forthcoming in *Research Policy*).

Ghosh, Rishab Aiyer. 1998.”Cooking-pot markets: An economic model for the trade in free goods and services in the Internet,” *First Monday*, 3(3). [Available at: http://www.firsmonday.org/issues/issue3_3/ghosh/index.html.]

______. 2003. “Clustering and Dependencies in Free/Open Software Development: Methodology and Preliminary Analysis,” MERIT-Infonomics Institute and SIEPR-Project NOSTRA Working Paper, 15th February). http://opensource.mit.edu/papers/ [Forthcoming in Joe Feller, Brian Fitzgerald, Scott Hissam, Karim Lakhani, eds., *Making Sense of the Bazaar*, Cambridge, MA: MIT Press, 2005]

Ghosh, Rishab Aiyer and Ved Prakash, Vipul. 2000. “The Orbiten Free Software Survey,” *First Monday*, 5:7. http://www.firstmonday.org/issues/issue5_7/ghosh/

Ghosh, Rishab Aiyer and Paul A. David. 2003. “The nature and composition of the Linux kernel developer community: a Dynamic Analysis,” SIEPR-Project NOSTRA Working Paper (21st February). http://opensource.mit.edu/papers/

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. http://www.infonomics.nl/FLOSS/report/

González-Barahona, Jesús M. et al. 2002. “Counting potatoes: The size of Debian 2.2,” (Version 3a: 3 January). http://people.debian.org/~jgb/debian-counting/counting-potatoes/.

Gonzalez-Barahona, Jesus and Gregorio Robles. 2003. “Free Software Engineering: A Field to Explore,” *Upgrade*, IV(4), August..

Gonzalez-Baharona, Jesus M., Luiz Lopez and Gregorio Robles. 2004. "The community structure of the modules in the Apache project." GSyC Working Paper, Universidad Rey Juan Carlos (Mostoles). February. http://opensource.mit.edu/papers/

Gonzalez-Barahona, Jesus and Gregorio Robles. 2004. “Getting the Global Picture,” A presentation at the Oxford Workshop on Libre Software (OWLS), Oxford Internet Institute, 25-26 June 2004. [Available at: http://www.oii.ox.ac.uk/fiveowlsgohoot/postevent/Barahona&Robles_OWLS-slides.pdf].

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

Kelty, Christopher M. 2001. “Free Software / Free Science.” *First Monday* 6 (12: December). www.firstmonday.org/issues/issue6_12/kelty/index.html.

Koch, S. and G. Schneider. 2000. “Results From Software Engineering Research Into Open Source Development Projects Using Public Data,” Vienna University of Economics and Business Administration http://opensource.mit.edu/papers/koch-ossoftwareengineering.pdf.

Kogut, B. and A. Metiu. 2001. “Open-Source Software Development and Distributed Innovation,” *Oxford Review of Economic Policy* 17 (2): 248-64.

Krishnamurthy, S. 2002. “Cave or Community? An Empirical Examination of 100 Mature Open Source Projects,” University of Washington, Bothell. (May). http://opensource.mit.edu/papers/krishnamurthy.pdf.

Kroah-Hartman, Greg, 2005. “How the Linux Kernel Development Process Works,” Groklaw, 29 May @2:45PM EDT. Available at: http://www.groklaw.net/article.php?story=20050529095918381.]

Lakhani, Karim and Eric von Hippel. 2000. “How Open Source Software Works: "Free" User-to-User Assistance.” Research Policy 32 (6): 923-943.

Lakhani, Karim R., Bob Wolf, Jeff Bates and Chris DiBona, 2003. “The Boston Consulting Group Hacker Survey (in cooperation with OSDN)”. [Available at: <http://www.osdn.com/bcg/bcg-0.73/BCGHackerSurveyv0-73.html >.]

Lehman, M. M. 1980. “Programs, Life Cycles and Laws of Software Evolution”, *Proc. IEEE*, 68,: pp. 1060-1078.

Lehman, M. M. 2000. “Rules and Tools for Software Evolution, Planning and Management,” in J. Ramil, ed., *Proc. FEAST 2000*, London: Imperial College of Science and Technology: pp. 53-68.

Lerner, Josh and Jean Tirole. 2002. “The Simple Economics of Open Source.” National Bureau of Economic Research (NBER) Working Paper 7600 (March). www.nber.org/papers/w7600.

Mateos-Garcia, J. and W. E. Steinmueller. 2003a. "The Open Source Way of Working: A New Paradigm for the Division of Labour in Software Development?" Falmer, UK, SPRU -- Science and Technology Policy Research, INK Open Source Working Paper No. 1. January.

——— 2003b. “Dynamic Features of Open Source Development Communities and Community Processes,” Brighton: SPRU -- Science and Technology Policy Studies, Open Source Movement Research INK Working Paper No. 3. February.

McGowan, D. 2005. “Legal Aspects of Free and Open Source Software,” Ch.24 in J. Feller et al., *Perspectives on Free and Open Source Software*, Cambridge MA: MIT Press.

McVoy, L., E. Raymond, et al., 1998. « A solution for growing pains (mailing list thread). Email to Kernel Mailing List (30 September). Available at: http://www.ussg.iu.edu/hypermail/linux/kernel/9809.3/0957.html.

Offer, Avner. 1997. “Between the Gift and the Market: The Economy of Regard”, *Economic History Review*, vol. 50, 3 (August): pp. 450-476.

Raymond, Eric. S. 1998. “The Cathedral and the Bazaar. *First Monday*, 3(3). Available at: http://www.firstmonday.org/issues/issue3_3/raymond.

______ 2001. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* Sebastopol, CA: O’Reilly.

Gregorio Robles, Juan Jose Amor, Jesus M. Gonzalez-Barahona, Israel Herraiz. 2005. “Evolution and Growth in Large Libre Software Projects”. In *Proceedings of the 8th International Workshop on Principles of Software Evolution*, September, 5-6 2005, Lisbon, Portugal. [Available at: http://gsyc.escet.urjc.es/~grex/iwepse05-robles-amor2.pdf.]

Robles, G., and J. M. Gonzalez Barahona, 2005, “Developer identification methods for integrated data from various sources,” Presented at *ICSE’05*, St. Louis, Missouri, May 15-21, 2005.

Robles, Gregorio, Stefan Koch and Jesus Gonzalez-Barahona. 2004. “Remote Analysis and Measurement by Means of the CVSAnalY Tool,” Working Paper, Informatics Department, Universidad Rey Juan Carlos, (June). [Available at: http://opensource.mit.edu/papers/robles-koch-barahona_cvsanaly.pdf.]

Robles, Gregorio, Juan Julien Merelo, and Jesus Gonzalez-Barahona, 2005.” Self-Organized Development in Libre Software: A Model Based on the Stigmery Concept,” In *Proceedings of the 6th International Conference on Software Modeling and Simulation*, St. Louis, Missiori. May.

Ross, Sheldon M. 2003 . *Introduction to Probability Models*. 8th Edition. New York: Academic Press.

Scacchi, Walter.2003. Understanding Software Evolution: Applying, Breaking, Rethinking the Laws of Software Evolution, Institute for Software Research-U.C. Irvine Working Paper. April. [Available at: http://opensource.mit.edu/papers/scacchi3.pdf.]

Simon, Herbert A. 1962. “The Architecture of Complexity.” Proceedings of the American Philosophical Society, 106 (December): 467-482.

Tuomi, Ilka. 2000. “Learning from Linux: Internet Innovation and the New Economy,” Working Paper (February) [available at: http://www.jrc.es/~tuomiil/articles/LearningFromLinux.pdf.]

_____ 2001.“Internet, Innovation, and Open Source: Actors in the Network,” *First Monday* 6(1), Jan. 8, 2001.

von Krogh, Georg, Sebastian Spaeth and Karim R. Lakhani. 2003. “Community, joining, and specialization in open source software innovation: a case study.” Research Policy, 32(7): 1217-1241

von Hippel, Eric. 2002. "Horizontal innovation networks - by and for users" Cambridge, MA, Massachussetts Institute of Technology, Sloan School of Management, Working Paper No. 4366-02. June.

von Hippel, Eric 2004. “Democratizing Innovation: The evolving phenomenon of user-innovation,”MIT Sloan School of Management Working Paper (November), [available at: http://advancingknowledge.com/drafts/VonHippel_Democratzing%20Innovation%20pap%20for%20Nik.doc].

von Hippel, Eric 2005. *Democratizing Innovation*. Cambridge MA: MIT Press.

Weber, Steven. 2004. *The Success of Open Source*. Cambridge MA: HBS Press.

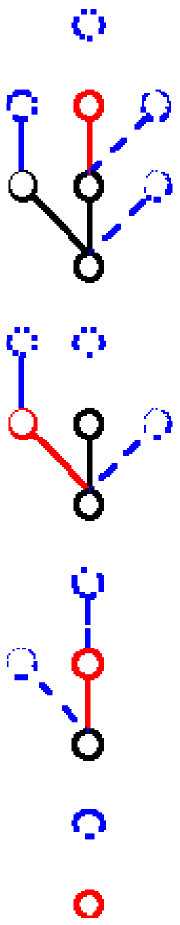


Figure 1

Graph representation of a software system growth process as an upward evolving tree

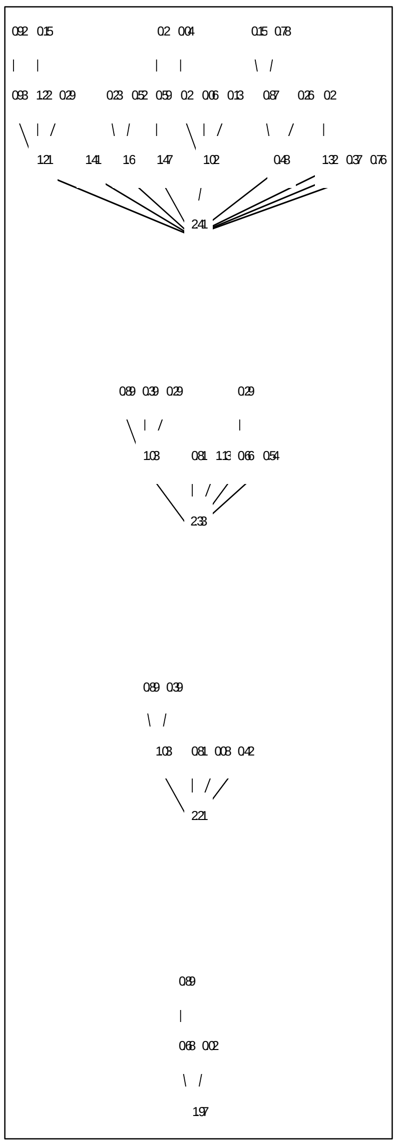
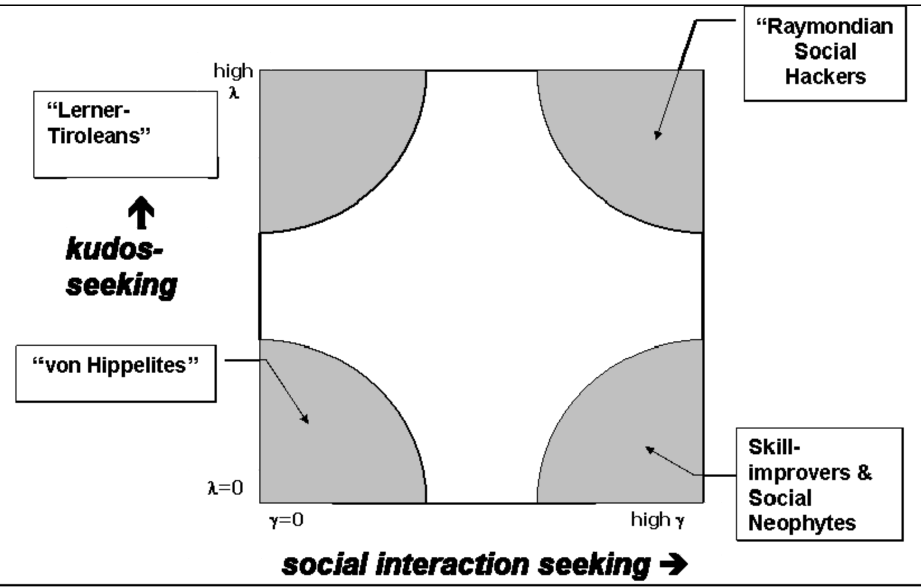


Figure 2

An illustrative simulation of the growth of a project's "code tree"


"Raymondian Social Hackers
high λ
"Lerner-Tiroleans"
kudos-seeking
"von Hippelites"
λ=0
γ=0
high γ
Skill-improvers & Social Neophytes
social interaction seeking

Representation of the Distribution of Salient Motivational Types among the Project's Contributors
Explanation: **The interpretation of the parameter pair (λ, γ) that most strongly affects the probabilistic positioning of sequential additions to the code-tree is that their configuration in the (λ, γ) plane reflects the underlying distribution of the four salient individual motivations for sustained participation – the pure types being located at the corners of the rectangular plane.**

Figure 3

***Figure 4*:**

Parameter Values Generating Linearity of Code Growth

Explanation: "R-square" is the mean value of the goodness of fit of a linear time-trend to the simulated growth paths of the project, given the indicated values of the parameter-pair (λ, γ).

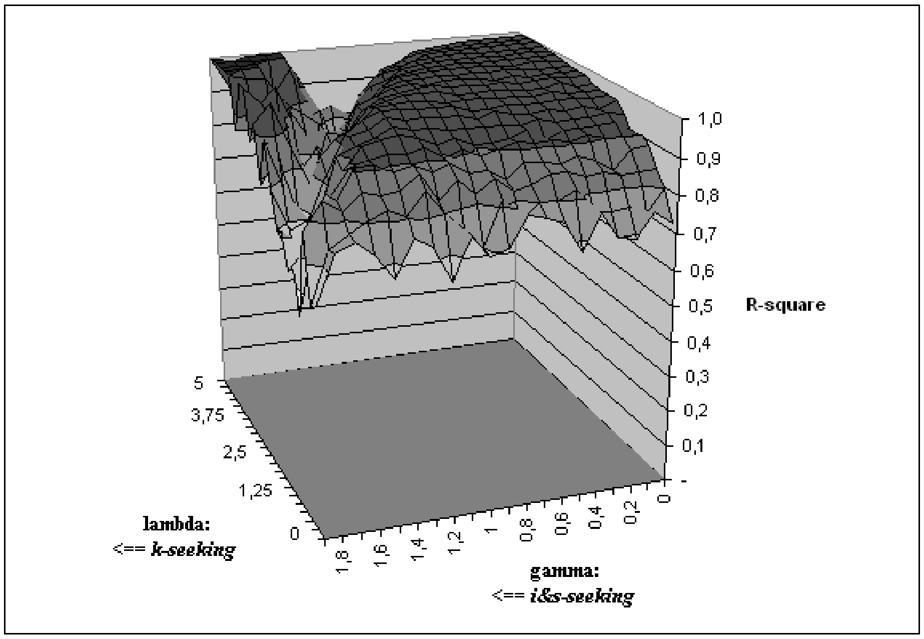
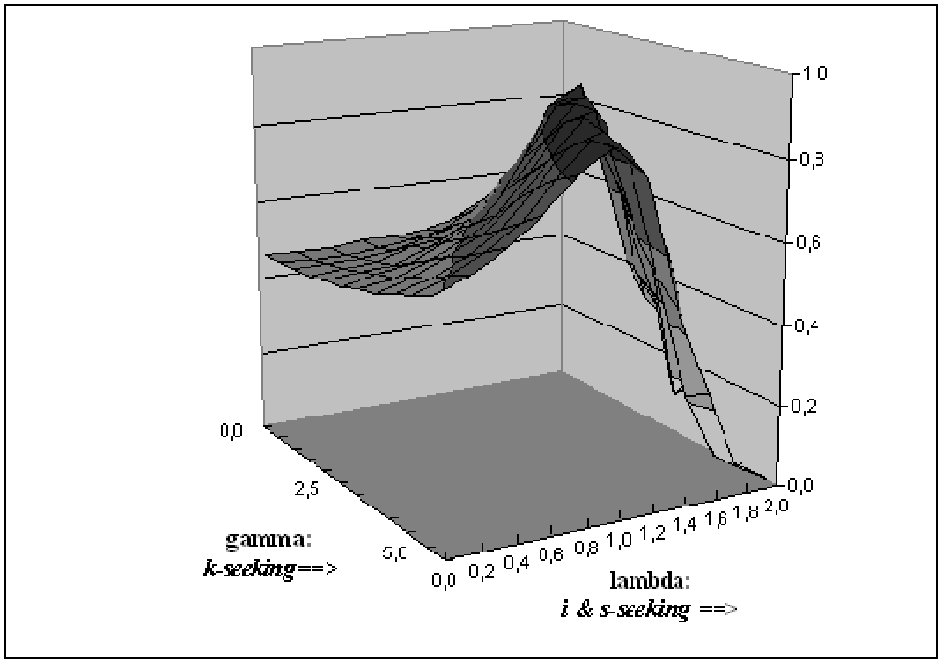


Figure 5

Gini coefficient for module size distribution



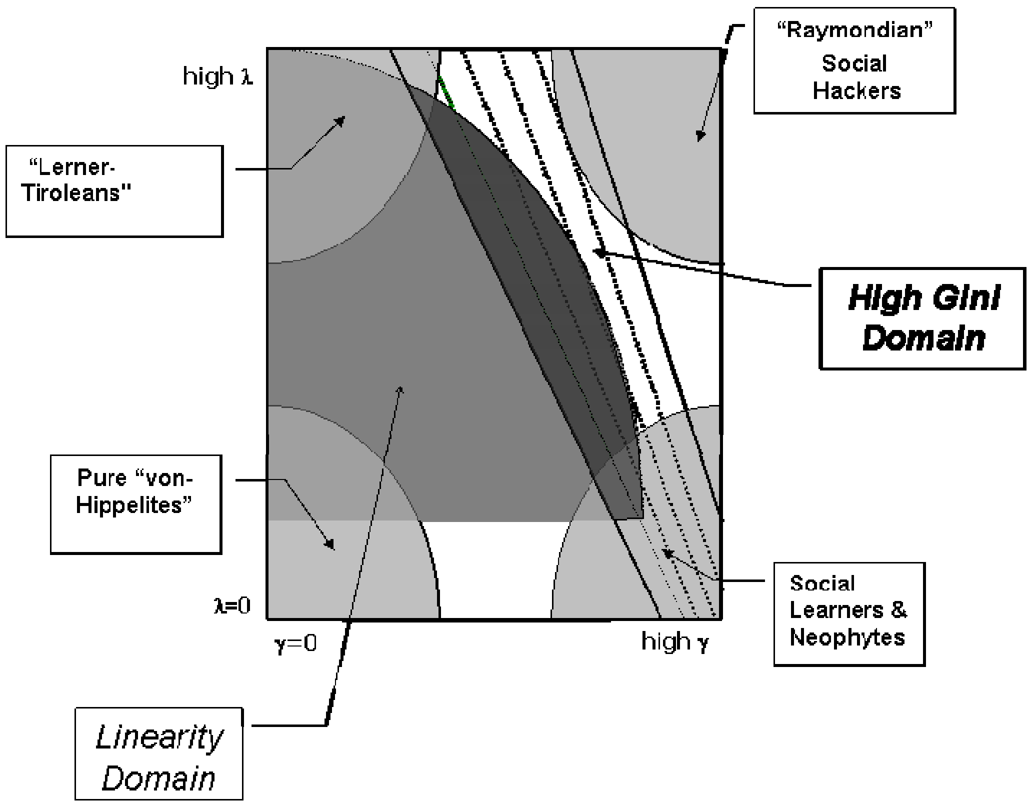


__Figure 6__

The location of the "empirically relevant zone" of the λ-γ parameter space

Explanation: The darkly shaded zone of the parameter-space (which generates linear growth in the number of project modules and high concentration in the distribution module code-size) is also the region away from all the corners: mixtures of all four "pure" motivation-types are necessary to characterize the expected responses of the ensemble of agents contributing code to the project.

Table 1: Distribution of Developers by "Motivation Profiles" Identified by Cluster Analysis of Sub-samples of the FLOSS-US Survey Respondents – Grouped by the known membership size of their current project

Clusters	I-Mode	C-Mode	Total	Characterization of cluster's motivation profile
Cluster 1 %	11 *5.14*	8 *4.55*	19 *4.87*	Non- ideological, expert, self- employed or company-sponsored to collaborate on FLOSS projects: **professionals.**
Cluster 2 %	12 ***5.61***	29 ***16.48***	41 *10.51*	*Didn't* need to modify existing code but like fixing bugs, learning new programs: **aspiring hackers.**
Cluster 3 %	102 *47.66*	79 *44.89*	181 *46.41*	Become better programmers, find out how programs work; work with like-minded, 'give back to community', support Libre ideology: **social learners.**
Cluster 4 %	26 ***12.15***	30 ***17.05***	56 *14.36*	Needed to modify existing code and fix bugs, see open source as the "best way": **experienced hackers.**
Cluster 5 %	63 ***29.44***	30 ***17.05***	93 *23.85*	Modifying existing software is *not* important, *nor* are learning, and interacting with like-minded others, wanted to 'give back to community', many launched their own projects: **(individualistic) user-innovators.**
Total %	214 *100*	176 *100*	390 *100*	

Source: Underlying data from David, Shapiro and Waterman (2006): I-Mode stands for "Independent Mode" (participant's known projects have 1 or 2 members) and C-Mode stands for "Community-Mode" (participant's known projects have 30 members or more).

Table 2: Module size distribution statistics for selected releases of the Linux kernel

Linux kernel release: date/ number	March 1994 1.0.3	April 1997 2.0.50	April 2002 2.5.25
Total bytes of code (millions)	4.5	21.0	133.8
Total physical lines of code (millions)	0.122	0.538	3.158
Modules (packages)	30	60	169
% of bytes in top 10% of modules	33.0	57.0	70.0
% bytes in top 20% of modules	51.0	74.0	79.0
Gini coefficient	0.54	0.68	0.72

Source: Ghosh and David (2003), Tables 1 and 6.