SIEPR Discussion Paper No. 04-02

SimCode: Agent-based Simulation Modelling of Open-Source Software Development

By
Jean-Michel Dalle
University Pierre-et-Marie-Curie & IMRI-Dauphine
And
Paul A. David
Stanford University & Oxford Internet Institute

November 1, 2004

Stanford Institute for Economic Policy Research
Stanford University
Stanford, CA 94305
(650) 725-1874

SimCode: Agent-based Simulation Modelling of Open-Source Software Development

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

Previous Draft: 5 September 2004

This version: 1 November 2004

ACKNOWLEDGEMENTS

The research reported in this paper was made possible by grant awards to the Stanford University (SIEPR) in support of the Project on the Economics of Free & Open Source Software (http://siepr.stanford.edu/programs/OpenSoftware_David/OS_Project_Funded_Announcmt.htm), from by the National Science Foundation program on Digital Technology and Society : IIS-0112962 (2001-04) and IIS-0329259 (2003-05). Excellent computational assistance in the simulation experiments was provided by Vincent Finet in the IMRI team at University of Paris-Dauphine. We have benefited also from our discussions of material in this paper with participants at the EPIP3 and OWLS seminars, which convened at Scuola Superiore Sant'Anna in Pisa, Italy, on 2-3 April 2004, and at the Oxford Internet Institute in Oxford, UK, on 25-26 June 2004, respectively. Fabio Arcangeli, Robin Cowan, Brian Fitzgerald, Alfonso Gambardella, Rishab A. Ghosh, Jesus Gonzales-Baharona, Bronwyn H. Hall, Jim Herbsleb, Eric von Hippel, Brian Kahin, Mathieu Lacage, Karim Lakhani, Juan Mateos-Garcia, Stephen M. Maurer, Jean-Charles Pomerol, Gregorio Robles, Walt Scacchi, and Ed Steinmuller all had a helpful hand in shaping this work, but the views expressed and the defects that remain are ours. Dalle wishes to dedicate this paper to the memory of his great uncle, who spent his life as a gardener, and who taught him most of the things he knows about growing trees.

ABSTRACT

We present an original modeling tool, which can be used to study the mechanisms by which free/libre and open source software developers' code-writing efforts are allocated within open source projects. It is first described analytically in a discrete choice framework, and then simulated using agent-based experiments. Contributions are added sequentially to either existing modules, or to create new modules out of existing ones: as a consequence, the global emerging architecture forms a hierarchical *tree*. Choices among modules reflect expectations of peer-regard, i.e. developers are more attracted a) to generic modules, b) to launching new ones, and c) to contributing their work to currently active development sites in the project. In this context, we are able – particularly by allowing for the attractiveness of "hot spots"-- to replicate the high degree of concentration (measured by Gini coefficients) in the distributions of modules sizes. The latter have been found by empirical studies to be a characteristic typical of the code of large projects, such as the Linux kernel. Introducing further a simple social utility function for evaluating the mophology of "software trees," it turns out that the hypothesized developers' incentive structure that generates high Gini coefficients is not particularly conducive to producing self-organized software code that yields high utility to end-users who want a large and diverse range of applications. Allowing for a simple governance mechanism by the introduction of maintenance rules reveals that "early release" rules can have a positive effect on the social utility rating of the resulting software trees.

SimCode: Agent-based Simulation Modelling of Open-Source Software Development

1. Introduction

By what mechanisms do free, libre and open source projects[1] mobilize their human resource inputs, allocate the contributors' diverse expertise, coordinate the participation and retain the commitment of their members? 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? How, in the absence of directly discernible market links between the producing entities and users or customers, is the output mix of the open source sector of the software industry determined?

To pursue answers to these questions resource allocation in open-source software one must enquired into the nature and functioning of behavioral and organizational mechanisms that, in effect, assign the (largely volunteered) efforts of members of a distributed community among a variety of distinct projects, and among the multiplicity of development tasks required by each of those projects. How then do developers collectively select among the observed array of projects that are launched? What processes govern the mobilization of resources sufficient to enable them to attain the stage of functionality and reliability that permits their being diffused into wider use – that is to say, use beyond the circle of programmers immediately engaged in the continuing development of the code itself?

Thinking about such issues is the economist's *métier*. Considered from that viewpoint, it is rather surprising that these key microeconomics questions remain virtually unasked, let alone unanswered in the already extensive and rapidly growing literature devoted to "the economics of open source software."[2] What is required, and has not yet been provided, is a representation of the variety of non-market mechanisms that can aggregate myriad decentralized, interdependent micro-level production decisions on the part of contributing (and non-contributing) agents, thereby generating an array of open-source products with particular functional properties and qualitative attributes that affect their attractiveness to users outside as well as within the project-community. Economists, of course, have ready to hand a paradigm of a decentralized system that will perform these allocation tasks: a system of markets for resource inputs and outputs that is guided by endogenously generated price-signals. But in the case at hand, one is confronted with a production and distribution system that appears to be "working" in the absence of both "commands" and price-signals. This may account for a good bit of the hesitancy that members of the discipline have displayed in pressing for answers to their standard questions about microeconomic resource allocation in this area.

---

[1] We will on occasion refer simply to "open source" code, without implying any distinction from the class of free and open source projects that release code under the GNU GPL. Although the term "libre source software" is more elegant in conveying the sense in which free and open source code is "free," it is less familiar in the English-speaking world.

[2] See, e.g., the listing of papers at: http://opensource.mit.edu/online_papers.php. Only a handful address themes of the kind described in the abstracts of Garzarelli and Gallopini (November 2003), and Dalle and David (2003).

In order to venture into this unfamiliar territory, and eventually to be in a position to exploit the abundance of data that is becoming more accessible for systematic empirical resource on open-source software development activities, clearly, there is a need some "special purpose" conceptual guides and a set of flexible tools suitable for exploratory analyses. We suggest that agent-based simulation modeling offers a method that can meet those requirements. Stochastic interaction models and their associated simulation experiments, such as the ones that are presented here, have long been found to be useful tools with which to gain insights into the properties of complex systems in which the endogenously governed actions of heterogeneous agents give rise to "emergent properties," collectively generated coherent outcomes whose eventuation cannot be discerned by understanding the motivational rules or "objective functions" of the individual actors (see e.g. Dalle, 1997). Among the developers engaged in open-source projects there are two main forms of interaction: they may communicate directly via email or other messaging systems, and they interact indirectly through the medium of the "code" that they contribute, use and improve upon. This suggests the two principal avenues along which it would be plausible to proceed in seeking to understand how coordination is achieved in these distributed systems of collective production. In the line of research on which this paper reports, we focus attention upon interactions among the ensemble of developers that respond to the evolving state of the source code, and hence indirect – being mediated entirely by the commonly observable state of the code itself.[3]

We may view the current status of the various pieces of code under development, which are the result of previous contributions – hence the importance of sequential interactions between them – as forming the set of "problems" that confront developers interested in a given project; among these problems individuals select which, if any, they will contribute to "solving." This formulates an important aspect of the collective coordination processes as a sequence of micro "problem choice" problems (Carayol & Dalle, 2000). The latter conceptualization can be made more tractable by setting to one side the programmer's "meta-choice" problem, that of deciding between working in the open source mode or becoming engaged in "close" software production activities undertaken by a commercially-oriented enterprise. The issues that have absorbed much of the attention of economists concerned with the open-source phenomenon -- questions of why people contribute their efforts to open-source projects, and why they do so not only on projects that will yield programs of a kind that are customized for their immediate use, but join large and more general purpose software system-building communities – will thus be set beyond the scope of this study.[4] This permits us to focus primary attention upon what has been described as

[3] Studies of both the direct and indirect interaction space would be complementary, and this is a directions in which it is hoped that quantitative research on open-source communities will proceed, e.g., as Baharona and Robles (2004) have proposed. Repositories of messages from email-lists maintained by open source projects, and other Internet-based communities provide the data sources on the structure of communications associated with the activities of these projects, and from which much could be learned regarding resource mobilization and task allocation to activities apart from writing and patching code. See Caldas (2002) for an analysis of clustering patterns and clique formation in the email communications of a scientific and technical work groups, which develops "web-metrics" techniques that have wider applications. With specific reference to open-source project communities, see the pioneering study by von Krogh, Spaeth and Lakhani (2003), which focused on the process through which new open-source recruits made the transitions to from passive subscription to a project's email list ("lurking") to active participation in discussions ("joining"), and then to actually submitting code.

[4] For survey findings on the diverse and changing motivations of participants in open-source development, which provide useful bounds for much of the theoretical conjecture by economists and other social scientists on that theme,

questions of "motivations at the margin:[5] issues not of why agents volunteer their efforts, but when and where they chose to put their efforts, and how much input is contributed to which tasks?

2. On simulation modeling, complexity and parsimony: "sculpting with Occam's Razor"

Before proceeding to take up those questions, a few additional remarks may be in order to underscore the case for creating simulation models as a means of gaining insight into the dynamics of software projection in the open-source mode, and the relevance of such investigations in the context of the broader problems pursued in several scientific disciplines, the most prominent at this point probably being economics, management, and software engineering. It is not really surprising that software engineering research has also started thinking that simulation tool would be relevant to support the production of new scientific knowledge (Ramil & Smith, 2003), although it seems that simulation in that context is understood as a way to address the sparseness of the certain kinds of empirical data of interest for the efficient engineering of complex code structures. We should emphasize that this is not the purpose for which we have turned to the simulation approach. On the very contrary, new empirical data is flowing almost every week about open-source software development, due to the renewed interested which this research area has recently granted, and one of the main issues here is more to integrate the various elements that are thus produced, to try to make them fit together, to possibly reject some of them as characterized by artifacts, or to ask for more detailed studies along the most promising lines.[6]

Of course, it is hoped that modeling and simulation tools like the one we are developing in the context of the SimCode project could also help in empirical studies, by allowing researchers, up to a point, to better test and report upon their hypothesis and conclusions in a framework where calibrated results from other teams would have been incorporated. To do this, however, we critically need an appropriately "open" methodology: a reasonably transparent simulation structure that others can elaborate and modify in order to address particular theoretical issues, or confront specific bodies of data. Thus, designing such a tool, we consider appropriate to make use of Occam's principle of parsimony as a methodology; put differently, we take Occam's razor a "meta-tool" for analytical tool-building. The reason for this has to do with the largely heuristic nature of the simulation tools that we need: were they aimed at being predictive, the use of Occam's razor would need to be questioned in light of all recent works about Kolmogorov's

---

see, Ghosh, Glott, Kreiger & Robles, 2002; David, Waterman & Arora, 2003; Lakhani, Wolf, Bates & DiBona, 2003.

[5] See Dalle, David, Ghosh and Wolak (2004) for further discussion of the differences and connections between the two classes of motivational questions that research on open-source projects should purpose; and an initial attempt to statistically identify the effective micro-level "signals" that affect effort allocation, and which are generated by the evolving structure of the code of the Linux kernel, and the cumulative distribution of contributions by developers among the evolving sub-projects, or modules, of the project.

[6] To appreciate the extent to which the development of tools for massive data-extraction and automated data-analysis of material from open source repositories has run ahead of conceptual frameworks and interpretative hypotheses regarding the processes generating the data, one has only to look at the group of papers posted during the summer of 2004 by Gonzalez-Baharona and his co-authors (Lopez, Ghosh and Robles) at: http://opensource.mit.edu/online_papers.php.

complexity and about similar statistical approaches vindicating or supporting Occam's razor. Here however, sculpting with Occam's razor is conceived as an individual and collective learning process where earlier simulation experiments give insights about further ones, i.e. about modifications and evolutions of the model itself.

One should think of designing a simulation tool in an environment where many researchers are producing empirical and theoretical knowledge about open-source software as a knowledge producing activity in itself. But it is one that is likely to proceed most quickly, and to prove most productive if parsimony places strong limits on the number of hypotheses and parameters that the model involves in any single formulation. Otherwise, users soon would be confronted with the proverbial "map that was as big as the country': an overly complicated model whose opacity would prevent the modeler, and other contributors, from readily isolating the main effects that underlay the features of the system's "emergent properties," and from learning what parameter values were required to mimic important features of the available data. Indeed, only a relatively simple simulation model is able to be correctly reviewed in a peer-reviewed journal – according to the authors' practical experience –, and only a relatively simple model in the spirit of Occam can be used by a scientific research community to test and experiment alternative pieces of working knowledge. Only by "sculpting with Occam's razor" is it really feasible to render the design and exercising of a stochastic simulation model an interactive process that helps the modelers heuristically select among various tentative hypotheses, and implement them to discover what implications they hold for observable distributions of variables describing the actually phenomenon of interest. This self-revealing approach avoids trapping the model-builder in sterile closed loops, because transparency and "simplicity" sets conditions encouraging openness in interactions with other researchers, and in the reception and integration of their (hopefully constructive) criticism.

In any case, we believe that modeling of this kind offers a framework upon which it will be possible to integrate empirical data about the extent and distribution of participation in open source program development, with observations concerning the social norms and organizational rules governing those activities. It thus takes a step beyond the preoccupation of much of the recent literature. Moreover, by facilitating investigation of the implications of the micro-behaviors among the participants in open source and free software communities, this modeling approach provides a powerful tool for identifying critical structural relationships and parameters that affect the emergent properties of the system.

Indeed, the tasks we have set for ourselves, and hopefully for some others, in regard to studying open source software represent an explicit attempt to the challenge of providing answers to the classic questions of whether and how this instance of a decentralized decision resource allocation process could achieve coherent and socially efficient outcomes. What makes this an especially interesting problem, of course, is the possibility of assessing the extent to which institutions of the kind that have emerged in the free software and open source movements are enabling them to accomplish that outcome – without help either from the "invisible hand" of the market mechanism driven by price signals, or the "visible hands" of centralized managerial hierarchies. Responding to this challenge requires that the analysis be directed towards ultimately providing a means of assessing the social optimality properties of the way "open source", "open science", and kindred cooperative communities organize the production and regulate the quality of the information tools and goods – outputs that will be used not only for their own, internal purposes, but by others with quite different purposes in the society at large.

In this paper, we report on the current status of project *SimCode*, as we have chosen to denote it – for obvious reasons. Section 2 presents an overview of the model and of the simulation tool we have developed, and suggests, passim, several items that should fit into the future research agenda that we and others should undertake; section 3 reports on simulation experiments, and Section 4 concludes.

3. The model[7]

3.1 Structure and rationale

The core of the stochastic simulation model of open source software production presented here is a behavioral kernel: heterogeneous developers face an existing set of software modules[8] – about the state of which we assume that they are fully informed[9] –, and they choose the module they will contribute to stochastically, according to their effort endowments and to the reward that each module can grant them. Heterogeneity, represented here by the existence of a stochastic (discrete) choice function, classically accounts for all the un-observed characteristics of each developer. Each developer will prefer to undertake the most rewarding tasks, according to a reward system still to be determined: however, this is not a deterministic choice as there are necessarily unobserved heterogeneous characteristics which drive this choice, and for which no model can account for if it wants to avoid to absolute contingency trap in which it would fall if it assumed it could take all relevant variables into account. A simple, and now relatively traditional way to handle this (Anderson, de Palma & Thisse, 1992), is to consider that the more rewarding modules will be chosen with a higher probability – or, in the statistical physicist's language now common in most disciplines including economics, to consider rewards as weights and to compute the probability that each module is chose according to a ratio between its weight and the sum of all weights, possibly distorted by various parameters and coefficients. Namely:

$$\mathbf{P}\left[\textit{chosen module} = (\textit{virtual})\ \textit{module } m\right] = \frac{\rho_m(\alpha)}{\sum_{i=1}^{\textit{all modules}} \rho_i(\alpha) + \sum_{i=1}^{\textit{all virtual modules}} \rho_{i'}(\alpha)} \quad (2.1)$$

Where $\rho_{\square}(\alpha)$ stands for the reward of α contributed to each module.

---

[7] The current version of this model, and its exposition, have enormously benefited from various comments and criticisms we have received from various people after we had previously opted for an "early" release (Dalle & David, 2003), precisely to elicit comments both from the academic community and also from participant observers in open-source projects. Any modelling exercise like this one implies some conscious level of abstraction and simplification: however, the modellers might not be immediately accurate in their modelling attempts, over-estimating some parameters while underestimating others, and therefore critically need insights and inputs from many other experts. Needless to say, this basic assumption still completely holds here.

[8] Which would probably correspond more to packages than to individual files according to the terminology in vigour in most open-source projects.

[9] Which implies that each new contribution is immediately made accessible to all developers. We do not account for now for the fact that some contributions are suitable not to be integrated in the code, depending notably on their relevance, and on maintenance policy, at least at the module level: see section 3 below for simulation experiments with various global maintenance rules.

Among un-observed characteristics, an important caveat concerns here the precise nature of the problems that each developer faces in its own idiosyncratic situation, a feature which is reportedly known as a significant determinant of developer choice among various open-source projects: from Eric S. Raymond's "Every good work of software starts by scratching a developer's personal itch." (Raymond, 1998a) to Eric von Hippel's user theory (Harhoff, Henkel & von Hippel, 2000; Franke & von Hippel, 2002; von Hippel, 2002) and to more recent and quantitative evaluations (see e.g. Ghosh, Glott, Kreiger & Robles, 2002; David, Waterman & Arora, 2003; Lakhani, Wolf, Bates & DiBona, 2003). This is something that we only deal with stochastically in the current version of the model: namely, we account for un-observed characteristics like this one, but we do not specify it fully yet. A later version of this model should involve the development of such an improved behavioral kernel, which would account for the matching process between developer and module characteristics – not underestimating the

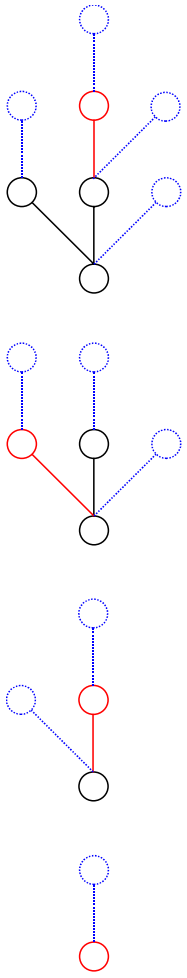

Figure 1

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

precautions that would be needed to support the validity of the claims then obtained as these developments would increase the non-ergodicity (David, 2001), and perhaps the deterministic features, of the system, due to a higher number of variables which would then accounted for.

It would mean developing a new module to the *SimCode* project. Indeed, this is exactly what open-source software development generally implies, since open-source developers do not simply consider adding their efforts to existing modules, but they also create new ones to supplement existing ones, when appropriate: this is precisely the mechanism that we have implemented to induce simulated code growth. To do so, we consider the following modeling finesse: we suppose that to each existing module 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 which would supplement this module: clearly then, the new module and the existing one would be technically linked[10], the new external module would typically be included in the existing during the compilation process, or sometimes simply called. 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.

In this framework, the emerging architecture of the modules is indeed mathematically a tree, since, by construction, there are no loops and each module is linked to only one (parent) module. This tree does not completely correspond to the actual directory tree, nor to the full set of technical and functional dependencies, which are usually known as the architecture of a software system per se (Bass, Clements & Kazman, 1998), since some of technical dependencies are not accounted for here, namely the fact that some modules can be called by several others. We have rather characterized it here as an emerging architecture, i.e. the one 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. Therefore, this emerging architecture here has much to do with the kind of phenomenon that Herbert A. Simon (1962) famously characterized years ago in a seminal article on the "architecture of complexity", and we indeed feel very much intellectually indebted to him, all the more so as the emerging architecture that he considers is also a tree-like "hierarchical system"[11]: Simon indeed precisely suggested that the emerging architecture of complex systems tended to often be spontaneously such, *because complex systems were born out of simple ones, and because simple systems then tend to be somehow included in more complex ones*. As for our modeling of open-source software development, the rationale for the emergence of a hierarchy of modules is strongly 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. We are also very close here to the recent research on modularity (Baldwin & Clark, 2000), and extending the model further in this

[10] Also in the sense of the wording of the GPL licence, for instance, which implies that if the "parent" module was GPL'd, then the new one would also be.

[11] But not in the traditional sense of hierarchy, just as a description of an architecture with several levels: indeed, so as to avoid mis-understanding, the French translation of this paper has precisely selected a word meaning "tree-like" (*arborescent*) to translate "hierarchy".

direction, notably by studying more extensively, and modeling more accurately, the actual technical interactions between modules, would also be a very fruitful research avenue

This model then allows us to test one of the main hypotheses that have been suggested about software development in open-source mode, namely, what we suggest to call the "regard" hypothesis. According to this theory, developers are significantly influenced by reputation effects: Eric S. Raymond (1998ab) was among the first to emphasize this idea in the famous essays he wrote as a participant observer in open-source communities; and it has been since suggested repeatedly by several other important studies of open-source software development, also as a more general attempt to analyze the striking similarities between open-source and open science communities (Benkler, 2001; Kelty, 2001; Dalle, David & Steinmuller, 2002)[12]. In a companion paper to this one (Dalle, David, Ghosh & Wolak, 2004), we indeed suggest that open-source software falls into a broader category which we characterize as peer regard economies: not reputation in a traditional sense, but rather to account for the fact that in these economies the actions undertaken by developers should *at the margin* account for the relative regard of their peers about their deeds[13].

We therefore suggest that developer statistically 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 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 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, our companion paper presents 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 (Dalle, David, Ghosh & Wolak, 2004).

Still according to the "regard" hypothesis, and to also account for other observations by Raymond and others, we also add the two following properties influencing developer choice:

[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.

This last property can be considered as a second-order effect, since it shows that developers and contributions are attracted by modules, which have already attracted more numerous

[12] On the economics of Open Science, see Dasgupta & David (1987, 1994), David (1998abc, 2000).

[13] At least when they are in C-mode, as opposed to I-mode: see Dalle & David (2003).

contributions, but it is also an important element of peer regard that contributions, however technically astute, should have an audience.

3.2. Mathematical description

In mathematical terms, we therefore get:

$$\forall m \text{ a module: } \rho_m(\alpha) = r_m(x_m + \alpha) - r_m(x_m) \qquad (2.2)$$

Where $\rho_m(\alpha)$ still stands for the expected[14] reward of contributing α to module m, $r_m(\square)$ 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 improvement of module m, i.e. precisely the sum of all past contributions, and α is a potential contribution for a developer's given effort endowment. Clearly then, by construction, for m' the virtual module associated with m:

$$\forall m': x_{m'} = 0 = r_{m'}(x_{m'}) \qquad (2.3)$$

Thus:

$$\forall m' \text{ the virtual module associated with module } m: \rho_{m'}(\alpha) = r_{m'}(\alpha) \qquad (2.4)$$

And $r_{\square}(\square)$ is a (positive) increasing convex function in coherence with rule [a] above, which imposes that the first contributions are more rewarded than the later ones.

We will further consider here that:

$$r_m(x_m) = r_{d(m)}(x_m) = v_{d(m)}(x_m)\, d(m)^{-\lambda} \left(\left(1 + c(m)\right)^{\gamma} \right) \qquad (2.5)$$

Where $d(m)$ is the distance of module m from the first "root" module; $v_{d(m)}(x_m)$ 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 $r_m(\square)$ into $r_{d(m)}(\square)$ is a direct consequence of the hierarchical and lexicographic assumption presented above: the reward associated with module m depends on its location in the software architecture only as it depends from the height of the module in the hierarchical module tree, $d(m)$. This dependency is then given according to characteristic exponent λ: when $\lambda = 0$ all modules are similarly rewarded, whatever their height $d(m)$, while as λ goes to infinity the dependency of rewards to the height of the module increases, with:

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

$$r_0\left(x_m\right) = v_0\left(x_m\right)\left(\left(1+c\left(m\right)\right)^{\gamma}\right) \text{ and } \forall m \neq root : r_m\left(x_m\right) \to 0 \text{ as } \lambda \to +\infty \qquad (2.6)$$

Since, by construction, the height of a virtual module is the height of its parent plus one, (2.4) and (2.5) above imply that:

$$\forall m : \rho_{m'}\left(\alpha\right) = r_{m'}\left(\alpha\right) = r_{d(m)+1}\left(\alpha\right) = v_{d(m)+1}\left(\alpha\right)\left(d\left(m\right)+1\right)^{-\lambda} \qquad (2.7)$$

If of course we note also that :

$$\forall m : \left(\left(1+c\left(m'\right)\right)^{\gamma}\right) = 1 \text{ since } \forall m : c\left(m'\right) = 0 \qquad (2.8)$$

By construction: the term in $c\left(m\right)$ in equation (2.5) above allows us to account for rule [b], namely, to render the more active projects – the "hot spots" – more rewarding for further contributions – even more and more so as γ increases, while 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.

We then define also:

$$v_{d(m)}\left(x_m\right) = \log\left(1+x_m d^{\mu}\right), \qquad (2.9)$$

where $\mu \geq 0$ is another characteristic exponent, which simply implies that it is easier to increase version numbers for high modules than for lower ones, and we easily verify that $v_{d(m)}\left(x_m\right)$, and therefore $r_{\square}\left(x_m\right)$ are positive increasing convex functions of x_m.

To complete the description of the model, what we are finally missing is a distribution of effort endowments α within the population of independent developers[15], normalized by individual productivities to directly translate into potential improvements added to modules: that is, a distribution of the size of contributions[16]. On the basis of the relative sizes of the high- and low-activity segments of the developer population found by various surveys, and notably the FLOSS survey, we suppose that these endowments are distributed according to an exponential distribution function[17]. Using the classical inverse transformation method on the cumulative

[15] Whatever their unit of measurement, typically in SLOC or in KLOC: if such a measure was to be selected, it should be noted that we do not differentiate here between lines added, replaced and deleted. As a consequence, a more appropriate measure of improvement would then be the sum of all lines added, replaced, and deleted.

[16] Since, as we mentioned above, this model is for now a model of contributions and not a model of contributors: the heterogeneity of contributions is a consequence of the heterogeneity of contributors, and we do not track for now for individual developers and for instance for the history of their contributions, which would necessarily imply to attach idiosyncratic characteristics to each individual developer. As a consequence, the model presented here is not properly speaking agent-based, but is more stochastic in its nature, accounting better for the intrinsic heterogeneity of economic actors through the observable heterogeneity of their actions.

[17] For now, we do not make any distinction different types of contributions, be they patches to correct bugs, or the addition of new features – which Raymond (1998a) indeed characterizes as the correction of "bugs of omission". This aspect of the model could certainly also be improved in later versions. We do not account either for the

distribution (e.g. Ross, 2003), we then compute the following exponential random number generator, which generates contributions α from a uniformly distributed probability:

$$\alpha = -\frac{1}{\delta}\ln(1-p), \qquad (2.10)$$

Where $p \in [0;1]$ is uniformly distributed and δ is a parameter which controls for the mean of the distribution, as a straightforward calculation will show that $\langle \alpha \rangle = \frac{1}{\delta}$, where $\langle . \rangle$ stands for the means.

Simulation experiments can then be run easily according to this model, in discrete time: at each time step a new contribution is simply added to the existing system[18], i.e. either an existing module is improved or a new one is created. The procedure is the following:

i. A random contribution is given by (2.10) ;

ii. The rewards of all existing modules, considering their current improvements, and of all virtual modules are computed according to (2.5), (2.7), and (2.9);

iii. A module is chosen according to (2.1), and the system is then modified in consequence.

Figure 2 represents the typical growth path of such a system, and should therefore be compared to Figure 1 above (numbers for each module are version numbers).

To finish with the mathematical description of the model, we just need to add that our ultimate goal is to analyze some of the characteristics of the emerging software systems, described here as code trees: in particular, we are interested in measure how sensitive their morphology (software-tree forms) is to parameter variation. Just to push the tree metaphor further, the obvious trade-offs of interest are those between intensive effort being allocated to the elaboration of a few "leaves," i.e. modules, which may be supposed to be highly reliable and fully elaborated software systems whose functions in each case are nonetheless quite specific, and the formation of a "dense canopy" containing a number and diversity of "leaves" that, typically, will be less fully developed. Indeed, a simple way to characterize this morphology, which we will use below, is simply to compute the Gini coefficient of the distributions of the sizes all "leaves" – modules.

The reason for this is that an important empirical finding, reported by Ghosh & David (2003), is that the Gini coefficients of the distribution of module sizes tend to be very (indeed, extremely) high. As for now, these results were obtained for the Linux kernel. This relatively striking feature means that there is a very limited number of modules with receive numerous contributions, and a very large number of modules with only a limited number of contributions, maybe only one[19].

---

involvement of commercial developers: we have started doing experiments in this respect, which will be reported elsewhere.

[18] As in all the experiments presented here, starting only with the root module with initial improvement 1.

[19] We would suggest that one-contribution (and therefore one-contributor) modules realized in I-mode can eventually be contributed to the project according to a global C-mode behavior (Dalle & David, 2003).

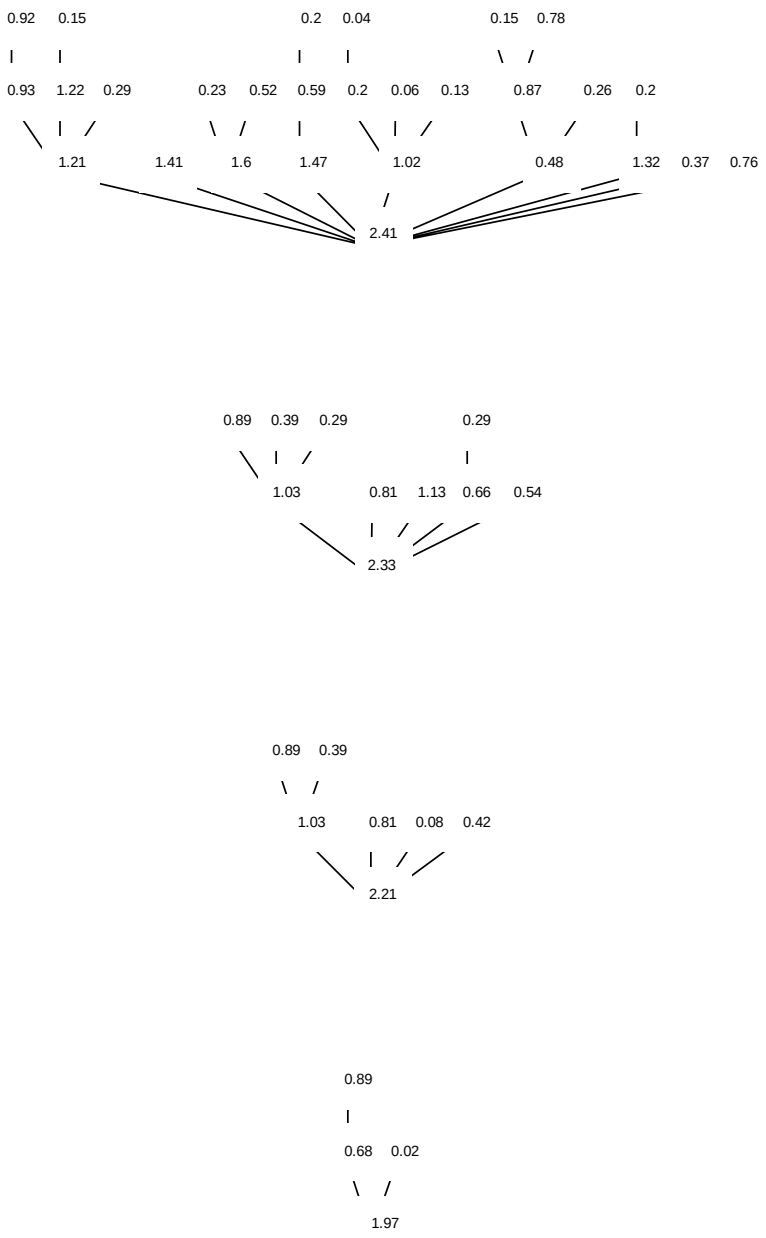

Figure 2

A simulation of the growth of a software project

But we also believe that the emerging morphologies of such software systems are absolutely non-neutral with regard to their social utility. Clearly, this should be and must be here a highly debated issue, but the reason why we dare enter this area is because we really feel critical to make some progress in the understanding of this difficult problem: for the purposes of this first step, the focus of the analysis is confined to showing the ways in which the specific norms of the reward system and organizational rules can shape emergent properties of software systems, such as its range of functions and reliability. Indeed, the global performance of development in open

source mode, in matching the functional and other characteristics of the variety of software systems that are produced with the needs of users in various sectors of the economy and polity, obviously, is a matter of considerable importance that will bear upon the long-term viability and growth of this mode of organizing production and distribution.

Therefore, we introduce here a simple social utility function, which basically captures 3 principles, which we first make clear:

(1) Lower modules are more socially valuable than higher ones because more users use them, and because also of the range of other modules and applications that eventually can be built upon them;

(2) A greater diversity of functionalities is more valuable because it provides software solutions to fit a wider array of user needs;

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

We then capture these ideas together according to the following[20] social utility function:

$$u = \sum_{m}^{(modules)} \left[\left(1 + v_d \left(m \right) \right)^{\nu} - 1 \right] d^{-\xi} \qquad (2.11)$$

Where $\nu \in [0;1]$ and $\xi \geq 0$ are parameters, again in the form of characteristic exponents: obviously, ν controls rule (3) above, while ξ controls rule (1) and while the summation in itself accounts for rule (2).

4. Simulating the allocation of efforts in open-source software development

For the sake of the exposition, Figure 3 (in annex) presents a typical collection of trees generated in the context of the simulation experiments presented in this section. In any case, all simulations reported in this paper have been conducted with similar values of the parameters, excluding λ and γ since they control the main regard effects that we intend to test, and which we will therefore keep as parameters, namely:

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

Which, at this point, can be considered as reasonable numerical values, all the more so similar results to the ones presented here hold for other values in the same range, except for higher

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

values of ξ which tend to eliminate the existence of non-corner maxima to social utility, by typically, and logically, driving the maxima toward low very high values of λ.

4.1 Simulation results on project architecture an the distribution of module sizes

Table 1 and Figure 4 now present Gini coefficients measuring the degree of concentration of the module-size distribution for various values of λ and γ, i.e. depending on the strength of the two main "regard" effects in the model: λ controlling the influence of the inner hierarchy of modules within the project, and γ controlling the attractivity of "hot spots" – active modules. Clearly, there is a region of the parameter-space within which both coefficients exert a positive influence on the Gini coefficient: one can see the boundary of that region describes a steeply rising "ridge-line" in Table 1 along which the entries for G attain a maximum in the neighborhood 0.86-.88. The row-maxima and column maxima for the Gini coefficient, which coincide along that ridge-line are marked in boldface in the table. In other words, there is a linear combination of λ and γ that consitutes a limit, above which the software system fails to develop, so that virtually all the code growth is confined to a single (root) module.[21]

We certainly do not generate Gini coefficients as high as those found in actual open-source project code (sometimes over 0.99), but this would have been impossible due to the limitations of our stylized simulation experiments; furthermore, we do not account for the technical peculiarities of some specific modules in a software project like Linux – where the modules providing a great variety of "drivers" results in a multiplicity of comparatively small code-packages, which contributes the projects high Gini coefficient. But simulations that displayed even the results did not hold in various simulation experiments that we conducted, and that are not described here in detail, for which Gini coefficients typically remained low (i.e., rarely exceedisng 0.5). This was for instance the case when we grew software systems:

i. Without rule [b] above, i.e. without the "hot spot" effect;

ii. Without rule [b], but with another rule, [c], accounting for a negative effect that a higher number of existing modules stemming from any given one would have on the motivation to create still another child to this module.

We would therefore suggest, according to these results but also to the other ones presented below which similarly exhibit high Gini coefficients, that there is a positive correlation between the existence of regard-based reward structures, and specially fashion effects, and the observed characteristics of package size distributions within some open-source software projects.

[21] A close approximation to this boundary line is found as: *max-Gini* = (0.1) λ + (0.43) γ. As one may see, this relationship begins to break down for value of $\lambda < 1$.

		γ										
		0,0	0,2	0,4	0,6	0,8	1,0	1,2	1,4	1,6	1,8	2,0
λ	0,0	0,47	0,47	0,47	0,48	0,47	0,48	0,50	0,55	0,61	0,70	0,68
	0,5	0,48	0,47	0,47	0,47	0,49	0,50	0,54	0,60	0,70	0,83	**0,73**
	1,0	0,48	0,48	0,48	0,49	0,51	0,53	0,58	0,67	0,82	**0,87**	0,67
	1,5	0,48	0,49	0,50	0,52	0,53	0,57	0,65	0,75	0,85	0,72	0,41
	2,0	0,50	0,50	0,51	0,54	0,58	0,61	0,74	0,86	**0,87**	0,71	0,32
	2,5	0,50	0,52	0,53	0,57	0,61	0,69	0,79	**0,88**	0,81	0,43	0,30
	3,0	0,52	0,53	0,56	0,60	0,65	0,74	0,85	0,87	0,61	0,38	0,17
	3,5	0,53	0,56	0,59	0,63	0,70	0,79	**0,88**	0,72	0,58	0,15	0,17
	4,0	0,55	0,57	0,61	0,66	0,75	0,84	0,86	0,61	0,56	0,22	0,05
	4,5	0,57	0,60	0,65	0,70	0,79	**0,87**	0,83	0,52	0,22	0,05	0,00
	5,0	0,59	0,62	0,67	0,74	0,82	**0,87**	0,79	0,46	0,25	0,05	0,00

Table 1: Gini coefficient for module size distribution (without maintainers)

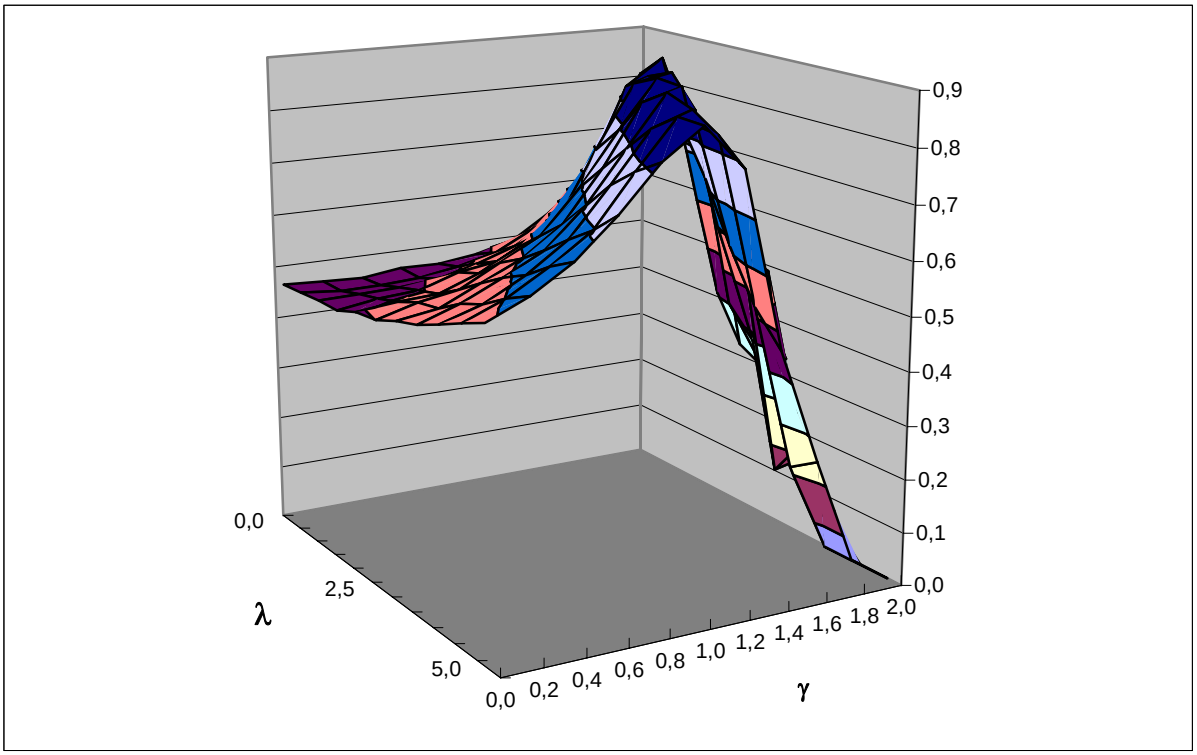

Figure 4: Gini coefficient for module size distribution (without maintainers)

4.2 Simulation results on developers' choices, project "release" rules, and social utility

To turn now to results about social utility, Table 2 and Figure 5 show that social utility varies systematically with λ and γ. But the effect of higher γ, raising the attractiveness of "hot spots" of developer activity among the modules, is to monotonically reduce the social utility of the overall project code. In Table 2 only the column maxima are marked in boldface, to highlight the fact that these occur at successively lower values of λ as the attractiveness of "hot spots" is increased, and that the value of the column maxima themselves decreases. It will be seen, therefore, that the locus of column maxima, and therefore the maxima of *social utility nowhere correspond* to the ridge-line region of Gini coefficients that appears in Table 1 and Figure 4 (Note that the γ axis has been inverted between the Figures 4 and 5, in order to obtain greater clarity in the perspective imposed by the 3-D view).

Although these results are remain quite tentative, it is difficult to escape the conclusion that the 'regard' motivations which we have hypothesized to operate within the open-source software communities of large projects are not conducive to generating socially optimal, or even second-best optimality in the emerging functional design of software systems. To put it differently, and still more hypothetically, if the motivations of independent developers drive them to take decentralized decisions that are responsive to "peer regard" and imitative of "social fashion" within the project-community (which would correspond to specifying parameters λ and γ in the "high Gini" zone), then the results could be considered as a less socially beneficial global outcome, compared to other situations were fashion and regard effects would typically have less potency in guiding developer's decisions. .

Needless to say, this rather striking conclusion rests entirely on the specification of the social welfare criterion, as well as the other behavioral specifications of the model. That it overturns the results reported by Dalle and David (2003) on the basis of an earlier version of the model is not problematic in itself: the present model, as has been seen, incorporated a previously overlooked "externality effect" – in the form of mimetic behaviors affecting individual choices about what parts of the project to which code is contributed, and effect that "social fashion – that enables the model to capture an important empirical feature of large projects' code, namely, the skewed distribution of the module sizes. But, there are still other empirical regularities, such the characteristics of the distribution of individual developer contributions to each of the modules, that the model in its still present, highly simplified form cannot simulate. Therefore, it is undoubtedly premature to attach any finality and certainly any policy significance to the finding just reported.

Nevertheless, in view of the importance and intrinsic theoretical interest of understanding the factors that will affect the assessment of open-source project performance from the viewpoint of external evaluators, and final users in particular –which our social welfare function seeks to represent, we believe it is appropriate to call attention to the foregoing results. At very least, it exposes the "instability" of the results yielded by the model during this still early phase of the sequential modification of its specifications. Indeed, one can do no less than report such reversals in results, if we are to adhere to the general scientific norm of "full disclosure" – placing one's trust in the latter's efficacy in promoting rapid, cumulative advances in knowledge.

		γ										
		0	0,2	0,4	0,6	0,8	1	1,2	1,4	1,6	1,8	2
λ	0	5,1	5,2	5,0	5,1	4,7	5,1	5,2	5,0	4,6	**4,3**	**2,4**
	0,5	7,1	6,3	7,0	6,8	6,7	6,4	7,0	5,6	**5,4**	3,5	2,3
	1,0	8,4	8,4	7,8	8,6	8,2	8,2	7,2	**6,5**	4,3	3,0	2,0
	1,5	10,0	9,5	10,1	9,5	9,4	8,8	**8,0**	6,2	3,4	2,1	1,7
	2,0	11,2	11,4	11,0	10,5	10,1	**9,2**	6,9	4,3	2,6	1,9	1,7
	2,5	12,3	12,1	11,4	11,2	10,2	8,6	6,4	3,2	2,2	1,7	1,7
	3.0	13,3	13,2	12,3	11,4	**10,5**	8,0	4,8	2,8	1,9	1,7	1,6
	3,5	13,8	13,4	12,4	**11,9**	9,6	7,0	3,7	2,0	1,8	1,6	1,6
	4.0	14,4	13,8	**12,6**	11,5	8,8	5,7	2,7	1,8	1,7	1,7	1,6
	4,5	14,6	**14,1**	12,4	10,6	7,8	4,6	2,4	1,8	1,6	1,6	1,6
	5.0	**15,0**	13,8	12,2	9,7	6,9	3,4	2,2	1,7	1,6	1,6	1,6

Table 2: Social utility (no maintenance)

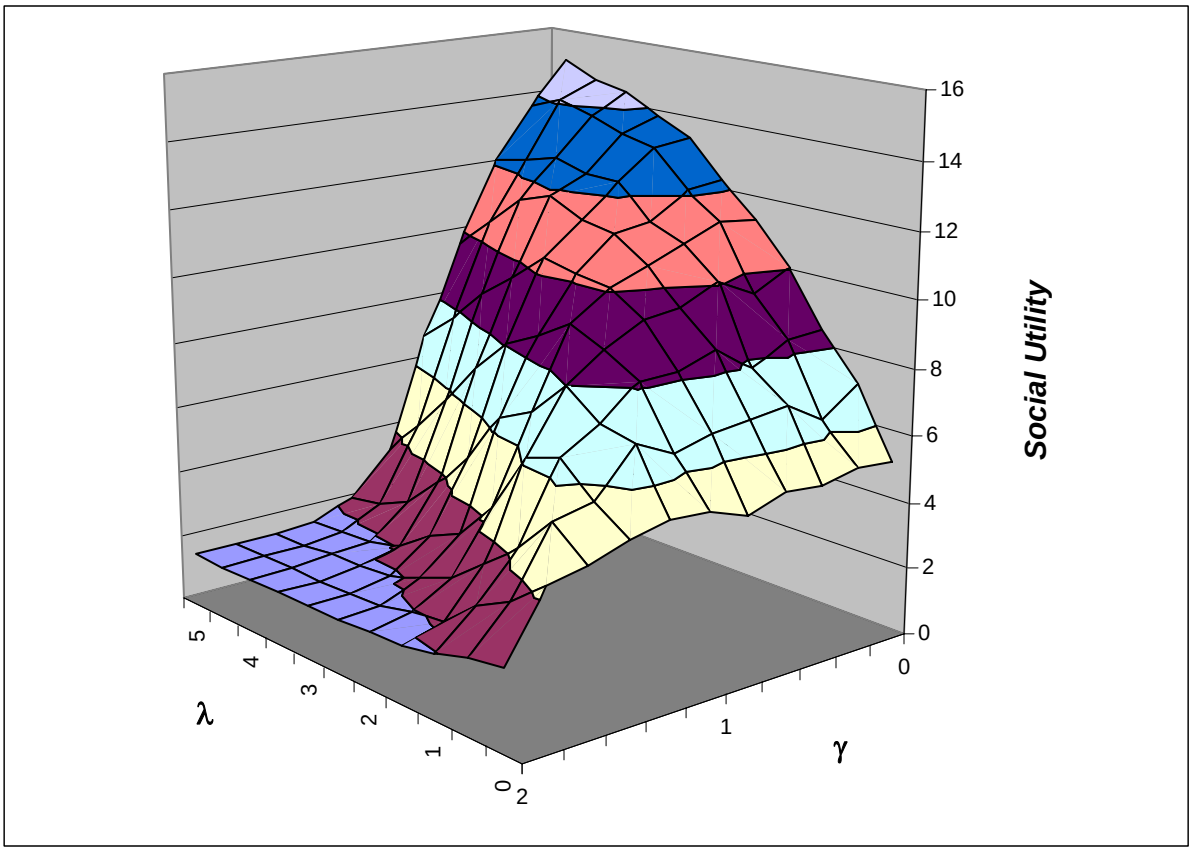

Figure 5: Social utility (without maintainers)

We have modeled here open-source software development in which the contributions are not selected, either ex ante or ex post, by maintenance criteria save for the endogenous regard norms that we have postulated for open-source development communities. This is clearly not so in most prominent open-source projects, including of course Linux and the crucial role still played by Linus Torvalds. Therefore, one may note that a simple (albeit highly stylized) way to introduce similar features into the present version of the model would be to provide one or more *maintenance rules*, according to which potential contributions to modules and/or virtual modules which do not follow these rules automatically receive a null reward. Consequently, such actions will never be chosen by the agents in the system, under the assumption that potential contributors know ex ante which maintenance rules will apply and have no (political) reason to offer code contributions that will not be accepted by the maintainers.

Stating the latter condition differently, this simple modeling finesse would abstract from the possibility of code-forking arising because developers were unhappy with the established maintenance rules and decided to continue contributing to a "variant tree." Such things do happen, but the phenomenon of deliberate code-forking driven by disaffection over maintainers' rules appears to be relatively rare in open-source development, and the current version of our model abstracts from it entirely. In a previous paper (Dalle & David, 2003), we had presented experiments realized according to such a rule, although we had then characterized it as a further norm, which the developers would follow, instead of considering it as a maintenance rule per se: but both situations are similar since we consider that all developer know and follow the maintenance rules. Namely, we had considered a "*limited tolerance to early release*" i.e.:

$$\forall m : r_{d(m)}(\alpha) = 0 \; iff \; v_{d(m)}(\alpha) \leq v_{\theta} \; ,$$

where v_{θ} is a version number release threshold below which contributions are not accepted (zero reward for developers). Although releasing early is often encouraged in open-source communities, it can be either tolerated to release projects largely before they are functioning correctly – developers then get credit for early releases –, or on the very contrary preference can be given to already functioning pieces of code. Our main finding was that early releases tended to raise the social utility of projects: according to our initial experiments, the lower the release threshold (even when it is down to 0), the higher then the social utility of the projects, and a tentative interpretation for this was that early enough releases created positive-sum games between the first and the next contributors. To put it differently, when the initial release was low enough, and considering the fact that early contributions are more rewarding, there was still sufficient esteem to be gained by contributing to the newly created module so as to attract further contributions, and compared to other opportunities like contributing to an already well developed module, or else creating yet another one.

Inasmuch as these results held without hot spot effects, which were not accounted for in Dalle & David (2003), Figure 6 presents similar results with $\gamma = 1.5$, which indeed hold for other values: we have selected here similar graphs to the ones presented in Dalle & David (2003), i.e. with $\delta = 3 \quad \mu = 0.5 \quad \nu = 0.5 \quad \xi = 2 \quad \lambda = 2$. These results tend to confirm that social utility still decreases as the release threshold is raised, although it now reaches a plateau on the right – once γ is so high that the software trees do not "develop" any higher level applications (i.e. the typical tree remains almost completely "stunted" at its root level).

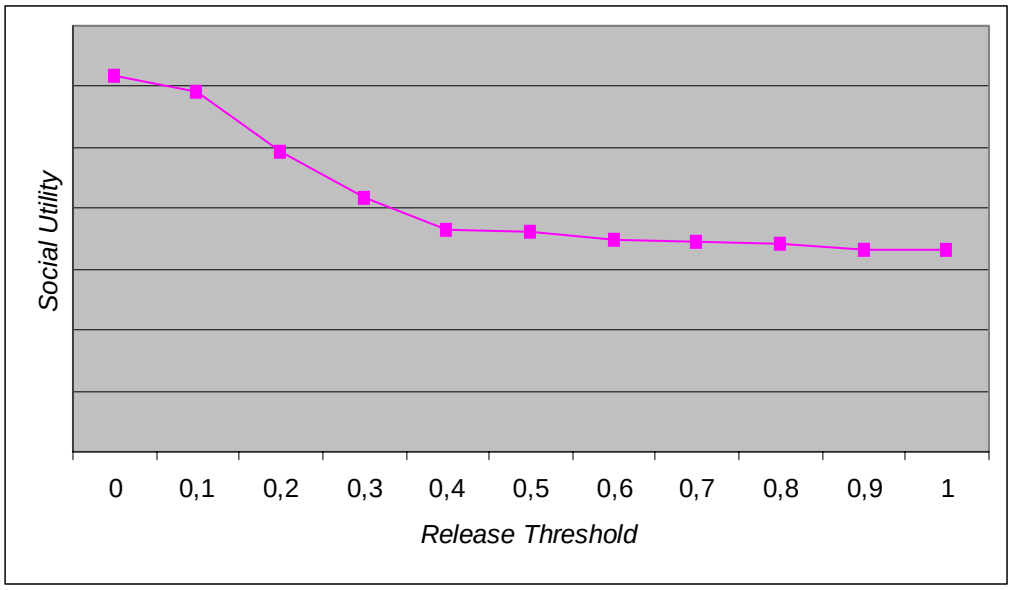

Figure 6: The impact of early release norms or maintenance rules

5. Conclusion

We have reported in this paper on the latest stage in the advance of project *SimCode*, an effort to construct a simulation model of software development in open-source mode. Obviously, the next steps can be taken in either of two directions. Following the empirical path and the iterative development, we can seek to calibrate the model more precisely by using the empirical regularities (e.g., on the types and sizes of modules, and the overall architectural morphology) observed in a number of large open-source projects of various kinds. But there are also clearly are a number of research agenda items in view on the analytical path, many having been set out by our first report on this undertaking (Dalle and David 2003), with several new ones being added in the course of the foregoing discussion. Perhaps the most important discrete elaboration will be the steps from modeling the tree to modeling a forest: adding typically a second "project tree" that may compete with the first for developers' contributions but also benefit from experience that they gain in working on the "rival" project. Next we envisage exploring whether the dynamics of the system becomes markedly more complex when the forest expands to beyond two trees, allowing some projects to have relationships marked by complementarity whereas other pairs are substitutes as far as the production relationships are concerned.

Looking ahead on both paths, it seemed obvious that it will be beyond our power to adequately pursue on our own even the principal items in the vast research agenda that we have opened – at least not at a rate that can keep up with the proliferating sources of empirical data that a fully specified model could illuminate, and 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 SimCode project in the distributed open-source manner: the next version of the model will provide not only the mathematical structure of a modularized version of the simulation structure, but will release the source code we are running, and which others may use to replicate our results and modify the structure. Whether this should formally become an experiment in the organizing of 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), is an intriguing question. But, for the present, the “open science” mode seems to be powerful and attractively familiar way in which to move forward – and invite others to join in the collective advancement of knowledge.

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.

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. “The Allocation of Software Development Resources in ‘Open Source’ Production Mode,” SIEPR-Project NOSTRA Working Paper, (15th February) [Accepted for publication in Joe Feller, Brian Fitzgerald, Scott Hissam, Karim Lakhani, eds., Making Sense of the Bazaar, forthcoming from MIT Press in 2004]

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

Dalle, Jean-Michel, David, Paul A., Rishab Aiyer Ghosh and Frank Wolak. 2004. “Free and Open Source Software Developers and the Economy of ‘Regard’: A Quantitative Analysis of Code-Signing Patterns within the Linux Kernel,” paper presented to the EPIP3 Seminar, Schuola Superiore Sant’Anna, Pisa, April, and to the Oxford Workshop on Libre Software (OWLS), Oxford Internet Institute, June.

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

Dalle, Jean-Michel and Nicolas Jullien. 2003. “ ‘Libre’ software : turning fads into institutions?”, *Research Policy*, 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. "Reputation and Agency in the Historical Emergence of the Institutions of ‘Open Science’," *Center for Economic Policy Research, Publication No. 261*, Stanford University, (revised March 1994), further revised :December.

——— 1998c. 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.

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.

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. 2003. “Clustering and Dependencies in Free/Open Software Development: Methodology and Preliminary Analysis,” MERIT-Infonomics Institute and SIEPR-Project NOSTRA Working Paper (First version: June 2002; revised: 15th February). [Accepted for publication in Joe Feller, Brian Fitzgerald, Scott Hissam, Karim Lakhani, eds., Making Sense of the Bazaar, forthcoming from MIT Press in 2004]

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).

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/

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.

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.

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 >.]

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.

Ramil Juan F. & Neil Smith. 2004. “Qualitative simulation of models of software evolution”, Journal of Software Process – Improvement and Practice, forthcoming.

Raymond, Eric S. 1998a. "The Cathedral and the Bazaar," *First Monday*, 3 (3: March), firstmonday.org/issues/issue3_3/raymond/index.html and www.tuxedo.org/~esr/writings/cathedral-bazaar.

———1998b. "Homesteading the Noosphere," *First Monday*, 3 (10: October), firstmonday.org/issues/issue3_10/raymond/index.html and www.tuxedo.org/~esr/writings/homesteading.

Ross, Sheldon M. 2003. “Introduction to Probability Models.” 8th Edition. Academic Press.

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

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.