Dynamics of Open Source Movements*

Susan Athey† and Glenn Ellison‡

January 2006

Abstract

This paper considers a dynamic model of the evolution of open source software projects, focusing on the evolution of quality, contributing programmers, and users who contribute customer support to other users. Our model has a public good problem that is partially mitigated by altruism. Programmers who have used features from open source software in the past are motivated to publish their own improvements, and the anticipation of these altruistic feelings may even lead them to choose to use open source rather than a commercial alternative that provides higher direct value. We consider two variants of the model, one in which programmer altruism is derived from the instrinsic quality improvement of the code, the other in which programmer altruism is related to the benefits end users derive from using the new code. In our model, end users require customer support to adopt open source, and this is provided by other users who received support themselves in the past. We show that to avoid a zero-quality steady state, projects require an initial critical mass of features and individuals willing to provide customer service. We derive additional comparative statics on the dynamics of this system. Finally, we analyze competition by commercial firms with OSS projects, showing that for many (but not all) parameter values, far-sighted commercial firms reduce their prices in order to slow the growth of OSS projects or even cause OSS projects to change trajectories towards the zero-quality steady-state.

*We are grateful to the Toulouse Network for Information Technology for financial support, and we thank participants in the 2005 Toulouse Network conference, in particular Josh Lerner and Jean Tirole, for helpful comments.

†Holbrook Working Professor of Economics, Department of Economics, Stanford University, Stanford, CA 94305. http://www.stanford.edu/~athey/, email: athey@stanford.edu.

‡Professor of Economics, Department of Economics, Massachusetts Institute of Technology, E52-380A, 50 Memorial Drive, Cambridge, MA 02138. http://econ-www.mit.edu/faculty/index.htm?prof_id=gellison, email: gellison@mit.edu.

1 Introduction

Open source software (OSS) has become a huge phenomenon, and it has recently gained attention in the economics literature (e.g. Lerner and Tirole (2005b)), the management literature (e.g. Shah (2006)), and the popular and trade press (e.g. Raymond (2001)). Although OSS has many varieties, common themes are that code is freely available, and contributions to the code are made by a diffuse set of mostly unpaid programmers working as volunteers. There are some well-known success stories in OSS, including Linux, which as of late 2005 had about 30% of the market for server operating systems, Apache, which dominates the market for internet servers, as well as PERL and PHP, which are market leaders in scripting software. Commercial firms often have complex relationships with OSS. Clearly, the competitors to OSS (notably Microsoft) are impacted by OSS, and competing with OSS products may be different than competing with traditional firms. OSS is also attracting a lot of investment by traditional firms; for example, IBM invested over $1 billion in open source initiatives in 2004.

Although much attention has been paid to the biggest projects, OSS is actually a much broader phenomenon. As of January, 2006, SourceForge.net claimed to host over 100,000 OSS projects and to have over a million registered users. A number of interesting questions arise about this population. Some OSS projects are long-lived, while others have a fairly short life-span. Some have grown quickly, while others stay small. The OSS projects differ in many dimensions. Of course, there is a wide range of products in terms of technical complexity, function, and scope. Other differences include: the size and composition of consumer base for products (end-users or developers/programmers), the mix of users and developers involved in the project, the governance structure, and participation by and funding from for-profit enterprises.

The goals of this paper are twofold. First, we seek to understand the dynamics of OSS projects–that is, changes in the size and composition of OSS projects over time–where the projects are considered in isolation. Second, we wish to analyze strategic pricing by a firm that faces an OSS competitor, when the OSS competitor is taken to be non-strategic. The results of our analysis can be used to answer questions such as: What is the role of the founders and early participants in an OSS project in terms of ensuring future growth? Is an initial critical mass of developers and end-users important in helping a project grow? Can strategic pricing by a commercial firm prevent OSS projects from attaining critical mass, or otherwise slow the growth of an OSS project?

Of course, the dynamics of OSS projects will depend on the motivations of the individuals who choose whether to use the product, contribute to the development and debugging of OSS, and provide customer support for other users. This paper takes these motivations as a behavioral primitive, rather than providing microfoundations derived, for example, on career concerns. The preferences we specify are consistent with much of the informal discussion of OSS (e.g. Lakhani and Wolf (2003), Shah (2004, 2006)).

In particular, we assume contributors are motivated by both their own needs and by a form of reciprocal altruism akin to that in Akerlof (1982) and Rabin (1993). Altruistic feelings arise when a programmer uses the product (something that is more likely when the quality is high), and it decays when the OSS fails to meets the need of a programmer. Without altruism, a programmer might develop a new feature but will not publish it and make it available to other users. The quality of the OSS develops over time, as programmers join the project and choose whether to contribute to it. Growth is then self-perpetuating, as high quality potentially generates more altruism; on the other hand, continual regeneration of the software is necessary as the quality naturally and exogenously depreciates over time. In our baseline model, contributors are the only players, and we consider how the dynamics and steady state qualities of OSS projects vary with factors such as the importance of altruism relative to the effort costs of publication and programming, as well as the depreciation rates of altruism and software features.

This simple model yields some interesting baseline predictions. First, there is always a steady state with zero quality and no altruistic programmers. However, so long as the flow of opportunities to add software and develop altruism are large relative to the depreciation of altruism and features, there is also a positive steady state. The dynamics of our simple model are somewhat different from what one might expect from informal discussions of open source software as being dependent on "network externalities." Starting from any initial condition with some features implemented in the software, the system converges to this the higher steady state. So in this baseline model, initial conditions and founder behavior have little to do with the long-run success of the project. However, the system requires both altruistic individuals and quality to grow; if a formerly commercial product becomes OSS, and there is no stock of altruistic programmers at the beginning, the quality of the product may fall for quite awhile until enough programmers join the community and gain altruism so that features are regenerated faster than they depreciate.

We turn next to consider competition between a commercial software firm and an OSS project. In particular, we consider optimal pricing by a single competitor to an OSS project.

We assume that the dynamics of OSS projects are determined by individual programmers acting independently as in the baseline model (rather than directed by a forward-looking, strategic leader). The optimal strategy of the commercial firm is given by the solution to a dynamic programming problem, which makes our model of competing with OSS somewhat different from that standard analyses of strategic interactions between competing firms. As long as the importance of altruism is not above a critical level, the commercial firm strategically prices below its static best response in an effort to slow the growth of the OSS project. However, some counter-intuitive results can emerge when altruism is very important and the altruistic programmer population is large. In particular, the commercial firm may price above the static optimum to speed the inevitable growth in open-source quality.

Clearly, the simple model misses some important features of OSS, such as the user base, and its predictions are in conflict with observations about the importance of initial conditions in getting an OSS project off the ground. Thus motivated, our next step is to extend the model to account for customer service. We introduce end users who do not contribute to the code, but who need customer support in order to use the code. We assume that these users are altruistic with some probability, so that they provide help if they received it themselves. We then extend the model further by allowing contributors to care about the number of users who will use the code in the near future. Customer support issues then become an important constraint on the growth of an OSS project, and it may be important that an initial set of committed programmers provide customer support early on until a large enough set of altruistic users is established to sustain customer support in the future. Otherwise, the project may collapse, as the lack of users inhibits the motivation of programmers to provide features, and the lack of features leads to a slow arrival rate of new customers who might provide customer service to the next generation of users.

Since initial growth is very important in the presence of customer support issues, a commercial competitor to an OSS project may have even greater incentives to price low in the early stages of an OSS project's lifecycle, in order to erode the project's user base and prevent it from reaching a critical growth level.

We conclude by describing several directions for future research.

A number of prior authors have written formal economic models of open source software. Lerner and Tirole (2002) model the incentives of open source contributors. They consider immediate and delayed benefits of contributing. Immediate benefits include monetary compensation (for contributors paid by other employers, or rarely, those employed

by the OSS project), own-use benefits, and the opportunity cost of time; long-term benefits include ego gratification from peer recognition and the more standard career concerns, since contributors may signal their ability to a wide community through OSS participation. Lerner and Tirole (2005a) consider the implications of career concern issues on the design of OSS licenses, and how the choice of licenses varies with characteristics of the project. They find suggestive evidence in favor of their theory using data from SourceForge.net.

Johnson (2004) compares the incentives for reporting errors within OSS and commercial products, hypothesizing that commercial projects create incentives for programmers to collude and suppress information about errors, since reporting errors may damage the reputation and career of the responsible programmer. He argues that the large number of individuals who can see and work with OSS code makes that type of collusion difficult to sustain in OSS projects.

Kuan (2001) considers a model of the OSS production function, where users contribute to the public good of the quality of the product. There are two types of users. Programmers contribute code up to the point that marginal effort in contributing equals the marginal cost of improving code (there are no costs of publishing after code is created), while users use an analogous calculation to determine their effort at reporting bugs. She provides empirical evidence comparing bugs for OSS and commercial products.

Johnson (2003) also analyzes OSS as a public good. The paper argues that a number of stylized facts about OSS can be understood by analyzing OSS through this lens, including things such as underprovision of documentation. More broadly, a wider user/developer base increases the quality of a project.

Like Kuan (2001) and Johnson (2003), our model incorporates the public good aspect of contributions to OSS. Programmers do not necessarily internalize the full benefits of publishing and sharing their code when they choose whether to write code, rather they consider the private benefits from using the code. However, we include altruism as a motive for publishing code, and we assume that generally, altruism is sufficient to outweigh publication costs, and may be large enough to induce some code to be written (in anticipation of future publication). In addition, our extended model of user-provided customer support also has a public goods problem, and again altruism serves as a partial but not full counterbalance to that.

Gaudeul (2004) takes a static or short-run approach to analyzing competition between OSS projects and commercial firms, focusing on the different types of features that may be developed by the two types of organizations in equilibrium. In OSS projects, a lack of

coordination leads to feature duplication or omission, and programmers do not develop a user-friendly interface. In contrast, the commercial firm may not find it worthwhile to pay programmers to implement all features. In equilibrium, the products are differentiated, and the OSS attracts customers who are either low-income or low-value, or else sophisticated developers who do not need a user-friendly interface.

Casadesus-Masanell and Ghemawat (forthcoming) present the only paper we are aware of that analyzes dynamic competition between OSS and commercial software. Their model takes as given that an OSS product exists and can commit to a price of zero for the product. Consumer demand for products is characterized by network externalities. The paper shows that the commercial product can avoid being pushed out of the market by forward-looking pricing policies, whereby the commercial firm always prices low enough to ensure itself a large enough installed base to ensure continued existence. Our approach is complementary to theirs, in that we allow for much richer dynamics in the OSS product, and we model the forces behind these dynamics. We do not, however, incorporate exogenous network externalities in the product market–instead, the size of the installed base affects the quality and viability of the OSS product through the creation, improvement and maintenance of the product itself. There are no network externalities for the commercial product in our model.

2 A Baseline Model

This section introduces our baseline model, where we focus only on software contributors (henceforth "programmers"). Consider a population of software programmers of unit mass. At Poisson random times each programmer is confronted a need drawn from a set of needs $N = [0, 1]$. Assume that the arrival times and the needs themselves are independent across programmers. Let λ be the parameter of the need arrival process.

At each time t the open source software package meets some subset $S_t \subset N$ of the needs. Write q_t for the Lebesgue measure of S_t. We'll refer to q_t as the quality of the software. The quality is a key outcome variable for the OSS project, and so we will be interested in how it evolves over time. In our baseline model, quality does not directly affect the set of programmers who consider using the product, although (as we see below) it indirectly affects the provision of new code through the encouragement of altruistic behavior.

Software programmers maximize lifetime discounted utility. Assume that an increment to utility is received whenever a need arises. The increment depends on the action taken by the programmer. Our assumptions about these increments are intended to capture

reciprocal altruism. Specifically, assume that the increment to programmer i's utility when he faces need n_{it} at t is:

B_{it}	if the need is met using the open source software (possible if $n_{it} \in S_t$);
$B_{it} - E$	if the need is met by programming;
$B_{it} - E - K + a_{it}$	if the need is met by programming *and* the programmer then adds the code to the open source project (possible if $n_{it} \notin S_t$);
B_0	if the need is instead met with an outside good.

The benefit B_{it} of meeting the need with open source software is assumed to be a random variable revealed to the programmer when he must decide on an action. The programming and sharing costs, E and K are assumed to be strictly positive. The altruism parameter $a_{it} \in \{0, a\}$ is stochastic and varies across programmers and over time. Assume that $\text{Prob}\{a_{it} = a | a_{it-dt} = 0\} = \alpha$ if programmer i meets his need using open source software at t. In intervals in which programmer i does not meet a need by open source altruism decays at a Posson random time, i.e. it follows a continuous time Markov process with $\text{Prob}\{a_{it+dt} = 0 | a_{it} = a\} = \delta dt$ and $\text{Prob}\{a_{it+dt} = a | a_{it} = 0\} = 0$.

The set of needs that can be met with the open source software grows when agents share code they have written. Assume that each feature of the software exogenously disappears at Poisson rate β. The motivation is that features become obsolete due to changes in interacting hardware and software.

We assume that agents can only observe aggregate behavior when they make their decisions. They understand the primitive parameters of the model, and they observe S_t (and thus q_t), as well as the realizations of random variables corresponding to their own outcomes. Whenever they are called on to take an action they myopically maximize the payoff their payoff from the current action.[1]

3 Developer Behavior and Community Dynamics

In this section we analyze the model described in Section 2.

[1] Given that agents only observe aggregates when making their decision the assumption of myopic play is similar to assuming that players play a sequential equilibrium of the dynamic game. The one difference is that a patient programmer would in some situations use open source software even though this is suboptimal in the short run, because he knows that will change his future utility function and allow him to receive the benefits that altruists receive when they behave altruistically. We do not think that this sophisticated behavior seems realistic.

3.1 Programmer Behavior

We begin with some fairly straightforward observations about programmer behavior in the baseline model. We organize the discussion by listing the observations as propositions.

Proposition 1 *If $a < K$ then no features are ever added to the open source software. Software quality decays at an exponential rate, $q_t = q_0 e^{-\beta t}$.*

Proposition 2 *Programmers use open source if it can meet their needs and $B_{it} > B_0$.*

This is an immediate consequence of the assumptions. Meeting the need by programming is dominated because $E > 0$. The 'program-and-contribute' option is only available if the feature is not already in the open source package.

Proposition 3 *Suppose that an programmer's need cannot be met by the open source software, that is, $n_{it} \notin S_t$. Then*
(a) If $a < K$ then the programmer develops the feature if $B_{it} > B_0 + E$.
(b) If $a > K$ then the programmer develops the feature and contributes it to the code base if $B_{it} > B_0 + E - (a - K)$.

A few comments about this proposition are in order. First, there is clearly a public goods problem. In the absence of altruism, programmers will develop features accounting only for their own private benefits, and they will never share their code after developing it (we have left out direct private benefits to publication, such as gaining future support and improvements for desired features, for simplicity). Second, altruism mitigates the public goods problem, and in fact there may be too much or too little development relative to the social optimum, depending on the magnitude of a. Altruism leads to strictly more features being developed if $a > K$. Programmers anticipate the utility they will gain from sharing the code (net of publication costs), and this offsets somewhat the private cost of effort. Indeed, agents may develop features where $B_{it} < B_0$ (no private benefits) if altruism is important enough.

To simplify the discussion in the remainder of the paper, we make the following assumption:

Assumption 1 *Assume $a > K$.*

It then follows immediately that:

Corollary 1 *Under Assumption 1 the equilibrium strategies are*

$$s_i^*(n_{it}; a_{it}) = \begin{cases} \textit{use open source} & \textit{if } n_{it} \in S_t \textit{ and } B_{it} > B_0, \\ \textit{program and contribute} & \textit{if } n_{it} \notin S_t,\ a_{it} = a,\ \textit{and } B_{it} > B_0 + E - (a - K), \\ \textit{program} & \textit{if } n_{it} \notin S_t,\ a_{it} = 0,\ \textit{and } B_{it} > B_0 + E, \\ \textit{use outside good} & \textit{if } n_{it} \notin S_t \textit{ and } B_{it} < B_0 + E - \min\{0, a - K\} \\ & \quad \textit{or } n_{it} \in S_t \textit{ and } B_{it} < B_0. \end{cases}$$

3.2 Dynamics

The status of the software and its future evolution is described by two state variables: the quality q_t of the software and the mass b_t of software programmers with $a_{it} = a$, *i.e.* the fraction who are currently altruistic.

We make the standard continuum-of-agents assumption that the law of large numbers holds exactly. We let γ_b denote the flow rate at which an programmer is confronted with a need for which an open source solution would dominate the outside option: $\gamma_b \equiv \lambda \text{Prob}\{B_{it} > B_0\}$. Similarly, we let γ_q denote the flow rate at which a programmer is confronted with a need that he would be willing to meet by programming and then contribute to the code if he were altruistic: $\gamma_q \equiv \lambda \text{Prob}\{B_{it} > B_0 + E - (a - K)\}$.

Proposition 4 *The dynamics of the system are given by*

$$\begin{aligned} \dot{q}_t &= \gamma_q(1 - q_t)b_t - \beta q_t \\ \dot{b}_t &= \alpha\gamma_b(1 - b_t)q_t - \delta b_t \end{aligned}$$

To gain some intuition for these dynamics, it is useful to begin by deriving the $\dot{b} = 0$ and $\dot{q} = 0$ curves. We begin by describing the $\dot{q} = 0$ curve. It is defined only for some values of q because when $q > \gamma_q/(\beta + \gamma_q)$ we have $\dot{q} < 0$ for any $b \in (0, 1)$. For $q \in [0, \gamma_q/(\beta + \gamma_q)]$ we let $b_{\dot{q}}(q)$ denote the value of b at which $\dot{q} = 0$, that is, the function defined implicitly by

$$\dot{q}(q, b_{\dot{q}}(q)) = 0,$$

so that

$$b_{\dot{q}}(q) = \frac{\beta q}{\gamma_q(1 - q)}.$$

Thus, the function is proportional to the ratio of the rate of decay of altruism and the arrival rate of programmer needs (β/γ_q) as well as the ratio of the fraction of features currently incorporated to the fraction of features that need to be written $(q/(1 - q))$. This implies some properties of $b_{\dot{q}}(\cdot)$ that will be useful for deriving steady states.

Proposition 5 *The function $b_{\dot{q}}(\cdot)$, which describes the curve along which quality is constant, is convex and strictly increasing on $(0, \gamma_q/(\beta+\gamma_q))$. It satisfies $b_{\dot{q}}(0) = 0$, $b_{\dot{q}}(\gamma_q/(\beta+\gamma_q)) = 1$, and $b'_{\dot{q}}(q) = \beta/\gamma_q(1-q)^2$.*

In words, $b_{\dot{q}}(\cdot)$ is increasing if for higher values of quality, a higher mass of software programmers must currently be altruistic in order for the quality of the software to remain constant over time. This follows as a result of our assumptions that quality naturally depreciates, and that new code is only published if a programmer develops the code and feels enough altruism to outweigh the publication costs. But if a feature currently exists, it won't be developed in the current time period, and so it won't be published. Thus, to avoid depreciation of quality, a high fraction of the programmers who develop the few remaining features must be altruistic and publish their code. Note that we could consider other models of the potential for quality improvements that do not have this "crowding" phenomenon; for example, in some settings, it might be that each new feature makes it possible for many more features to "build on it" and expand the appeal of the product in new directions.

The function $b_{\dot{b}}(\cdot)$ is convex if the rate of change in the stock of altruism necessary to compensate for an increase in quality in order to hold quality constant is higher for higher levels of quality. This again follows directly from our assumptions about how potential quality improvements relate to the current set of features of the product.

Now we turn to consider how the set of altruistic programmers must change with the quality of the product in order to keep the fraction of altruistic programmers constant. Observe that $\dot{b} = 0$ if and only if

$$\alpha\gamma_b q = (\alpha\gamma_b q + \delta)b.$$

For a given q, let $b_{\dot{b}}(q)$ denote the value of b at which $\dot{b} = 0$, that is, the function defined implicitly by

$$\dot{b}(q, b_{\dot{b}}(q)) = 0.$$

Then,

$$b_{\dot{b}}(q) = \frac{\alpha\gamma_b q}{\alpha\gamma_b q + \delta}.$$

Proposition 6 *The function $b_{\dot{b}}(\cdot)$ is concave and strictly increasing on $(0,1)$. It satisfies $b_{\dot{b}}(0) = 0$, $b_{\dot{b}}(1) \in (0,1)$, and $b'_{\dot{b}}(q) = \delta\alpha\gamma_b/(\alpha\gamma_b q + \delta)^2$.*

In words, $b_{\dot{q}}(\cdot)$ is increasing if for higher values of quality, a higher fraction of software programmers must currently be altruistic in order for the set of altruistic programmers to remain constant over time.

The function is concave if this effect is less pronouced at higher levels of quality. Note that the system has $\dot{b} > 0$ when $b < b_{\dot{b}}(q)$ and $\dot{b} < 0$ when $b > b_{\dot{b}}(q)$, so one can think of b as evolving toward this curve.

The system is in steady state where $b_{\dot{q}}(\cdot)$ and $b_{\dot{b}}(\cdot)$ intersect. Note that the two curves always intersect at $b = q = 0$. Hence, $(0, 0)$ is always a steady state of the system. The full behavior of the system follows fairly simply from the properties noted in the two propositions. Essentially, there are only two possibilities as pictured in Figure 1 below, which graphs the $b_{\dot{b}}$ and $b_{\dot{q}}$ curves in q-b space. The $b_{\dot{b}}$ curve is concave and the $b_{\dot{q}}$ is convex. If the $b_{\dot{q}}$ curve is steeper at the origin, the two curves will have no intersections other than at $(0, 0)$, as in the panel on the left. If the $b_{\dot{b}}$ curve is steeper at the origin, then the fact that the $b_{\dot{b}}$ curve intersects the right side of the square (i.e. $b_{\dot{b}}(1) \in (0, 1)$) and the $b_{\dot{q}}$ curve intersects the top side of the square implies that there is an unique interior intersection. The right panel illustrates such a system.

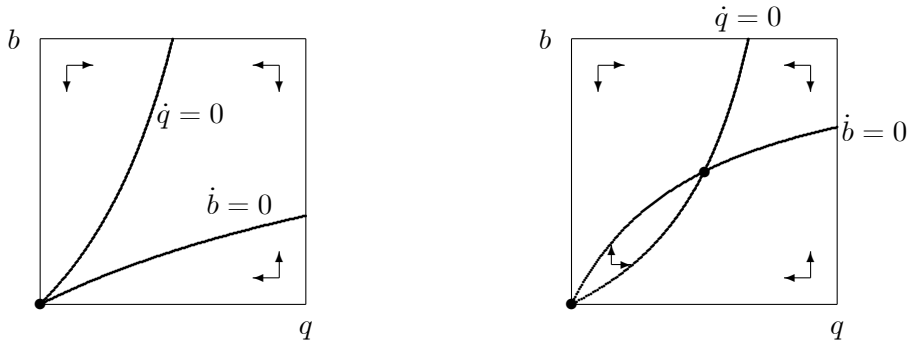

Figure 1: Model Dynamics: the left panel has $p = 0.5$ and $\gamma_b = \gamma_q = \beta = \delta = 1$. The right panel has $p = \gamma_b = \gamma_q = 1$ and$\beta = \delta = 0.5$.

Evaluating the derivatives of the two curves at the origin we have.

$$b'_{\dot{b}}(0) = \frac{\alpha\gamma_b}{\delta}$$
$$b'_{\dot{q}}(0) = \frac{\beta}{\gamma_q}$$

Proposition 7 *If $\alpha\gamma_b\gamma_q < \beta\delta$ then the only steady-state of the system is $q = b = 0$.*

If $\alpha\gamma_b\gamma_q > \beta\delta$ then the model also has a second steady-state with q and b positive.

The condition that $\alpha\gamma_b\gamma_q > \beta\delta$ has a very straightforward interpretation: the opportunities to add software and feelings of altruism occur sufficiently often relative to the speed at which altruism and features depreciate. Note that the depreciation of altruism and the depreciation of features enter symmetrically.

Solving for the positive steady state state we find:

Proposition 8 *If $\alpha\gamma_b\gamma_q > \beta\delta$ then the nonzero steady state of the model is*

$$\begin{aligned} q^* &= \frac{\alpha\gamma_b\gamma_q - \beta\delta}{\alpha\gamma_b\gamma_q + \alpha\beta\gamma_b} \\ b^* &= \frac{\alpha\gamma_b\gamma_q - \beta\delta}{\alpha\gamma_b\gamma_q + \delta\gamma_q} \end{aligned}$$

From here, we see that the steady-size of the project (and of the altuistic community) is increasing in $\alpha\gamma_b\gamma_q - \beta\delta$, the gap between opporunities to add software and develop altruism relative to the depreciation of features and altruism. We can also compare the growth rate of quality to the growth rate of altruistic programmers, finding that it depends on the decay rates. In some extreme cases, we get clear answers: when the decay rate of altruism is low ($\delta \approx 0$) we that the steady state fraction of altruistic programmers is 1 ($b^* \approx 1$), while when the decay rate of features is close to zero ($\beta \approx 0$), we get a steady state quality of 1 ($q^* \approx 1$).

Simple inspection of the phase diagram for the system leads to the following conclusions.

Proposition 9 *If $\alpha\gamma_b\gamma_q < \beta\delta$ then the steady state at $(0,0)$ is globally stable.*

If $\alpha\gamma_b\gamma_q > \beta\delta$ then the system converges to the (q^*, b^*) steady-state from every initial condition other than $(q_0, b_0) = (0,0)$.

This result implies that the dynamics of the system are deterministic and do not have history-dependence. Any project that gets off the ground with a few features or committed (altruistic) programmers will eventually reach a steady state that is predetermined given parameters. It can also be shown that projects tend to grow with quality and proportion of altruistic programmers roughly proportional to the steady state values all along the way.

Finally, we observe that the system can exhibit some nonmonotone behavior if we start from a skewed initial condition. For example, if some formerly commercial software is made public and thereby starts with q large and b small, then q may drop for a long time and become quite low before b catches up and allows quality to increase back toward the steady-state level.

4 Competing with Open Source

An important question for public and business policy concerns how competition between an OSS product and a commercial product differs from competition between two commercial products, or from monopoly pricing. Consider a single commercial software product competing with a single OSS project. We can incorporate this into the model of Section 2 by assuming that the "outside option" that provides utility B_0 is a choice between two goods: the commercial software that provides utility $v - p$, where v is the customer value of the commercial software and p is its price; and ignoring the need, which provides utility 0. For simplicity, we maintain the assumption that the commercial software can meet all needs and that all consumers have the same value for the commercial software.

Our model abstracts from direct network effects of the form analyzed by Casadesus-Masanell and Ghemawat (forthcoming), described in the introduction. The only network effects in our model come through the provision of features by altruistic programmers, and how that depends on the past actions of other programmers.

If $v < p$, the commercial firm gets no demand. For $v > p$, the commercial firm's demand comes from:

1. Programmers whose needs could be met by open source but have $B_{it} \leq v - p$;
2. Programmers whose needs cannot be met by open source, who are not altruistic, and who have $B_{it} - E \leq v - p$; and
3. Programmers whose needs cannot be met by open source, who are altruistic, and who have $B_{it} - E + (a - K) \leq v - p$.

Suppose that B_{it} has CDF G. Assume that the commercial has zero costs. Then, its flow profit function as a function of the quality of the OSS, the set of altruistic programmers in the OSS, and the price p of the commercial product (assuming $v < p) is s \pi(p; q, b) = p\left(qG(v-p) + (1-q)(1-b)G(v-p+E) + (1-q)bG(v-p+E-(a-K))\right).$

A basic observation on the form of this profit function is:

Proposition 10 *Flow profits are decreasing in the fraction b of programmers who are altruistic toward the open source project.*

Flow profits are decreasing in the quality q of the open source project if altruistic programmers are not too altruistic $a - K \leq E$, but otherwise will not be monotonically decreasing in q.

The reason why profits can be increasing in q is that programmers do not get the altruism benefit if they add a feature to the open source project that is already there. Hence, a package lacking a feature can be more attractive than a package with the feature. When b is large, this effect dominates.

4.1 Static profit maximization

Consider first the static profit maximization problem:

$$\max_{p:p\leq v} \; p\left(qG(v-p) + (1-q)(1-b)G(v-p+E) + (1-q)bG(v-p+E-(a-K))\right). \quad (1)$$

Note that it is of the form

$$\max_{p} p\left(\sum_{j=1}^{3} d_i G(\hat{v}_i - p)\right), \quad (2)$$

with $d_1 + d_2 + d_3 = 1$, where d_i is the fraction of total firm consumers coming from $i \in \{1, 2, 3\}$, corresponding to the three groups of consumers described above, and $\hat{v}_i$ is the net benefit to the consumer of type i from using the commercial product rather than the OSS at zero price. The first-order condition for such a problem is

$$\sum_j d_j G(\hat{v}_j - p) - p\sum d_j g(\hat{v}_j - p) = 0,$$

which gives

$$p = \frac{\sum_j d_j G(\hat{v}_j - p)}{\sum_j d_j g(\hat{v}_j - p)} = \frac{Q(p)}{\sum_j d_j g(\hat{v}_j - p)}.$$

One case in which this expression takes a very simple form is if the distribution of B_{it} is uniform on $[0, \overline{v}]$ for $\overline{v} > v + E$. In this case, the solution reduces to

$$p = \sum_j d_j \hat{v}_j / 2,$$

yielding

$$p^*(q, b) = \frac{1}{2}\left(v + (1-q)E - (1-q)b(a-K)\right).$$

The maximum $p^* = (v + E)/2$ occurs when $q = b = 0$. It has $p^* = v/2$ independent of b whenever $q = 1$. The price when $q = 0$ and $b = 1$ is $(v - E - (a - K))/2$. Note that if E is large enough, these calculations could yield $p > v$ which cannot be optimal; in such cases the firm chooses $p = v$.

Monopoly pricing in the absence of an OSS competitor is very simple in this example: the firm charges $p = v$. Unsurprisingly, the presence of a competitor reduces the optimal price. For the uniform case, we see that how the static optimum changes with the quality of the OSS depends on parameters:

$$\frac{\partial}{\partial q} p^*(q, b) = b(a - K) - E.$$

If programming effort is small relative to altruism benefits (weighted by the number of altruistic programmers), the commercial firm actually increases its price in response to a higher quality competitor. Otherwise (and always when $a - K < E$), we obtain the more intuitive result that a higher quality OSS product leads to a lower best response price by the commercial firm. It is also straightforward to see that the more altruistic programmers there are, the lower the optimal price of the commercial product.

4.2 Dynamic profit maximization

The dynamic profit-maximization problem for the firm is

$$\max_{p(q,b)} \int_{t=0}^{\infty} \pi(p(q_t, b_t); q_t, b_t) e^{-rt} dt \tag{3}$$

subject to

$$\begin{aligned} \dot{q} &= \lambda(1 - q)b\,(1 - G(v - p + E - (a - s))) - \beta q \\ \dot{b} &= p\lambda q(1 - b)\,(1 - G(v - p)) - \delta b, \end{aligned}$$

where the latter two equations are the laws of motion for the OSS quality and the number of altruistic programmers, given their choices between OSS and the commercial product. Note that the programmers are not forward-looking in this model, but rather make myopic choices based on the flow benefits of programming, publishing, or using the commercial product.

The dynamic problem is pretty straightforward when the altruism parameter is not too large: $a - K < E$. In this case, flow profits are decreasing in both q and b. Lowering p

decreases both $\dot{q}$ and $\dot{b}$. This plus the montonicity of the (q, b) system implies that the firm will always choose prices that are below the static profit-maximizing levels.

Note that $\frac{d\dot{q}}{dp}$ and $\frac{d\dot{b}}{dp}$ are both zero at $(q, b) = (0, 0)$ and at $(q, b) = (1, 1)$. This may imply that the departure from static profit maximization goes to zero at both of these points. If it does, the system should continue to have an interior steady state under dynamic profit maximization when it has an interior steady state under static profit maximization (because the (q, b) dynamics will still be upward in a neighborhood of $(0, 0)$).

The behavior of the system when $a - K > E$ is less clear. In this case profits are increasing in q when b is large (recall the discussion from the last subsection). Choosing a higher p increases $\dot{q}$ (although it also increases $\dot{b}$). It is therefore possible that prices will be distorted upward relative to the static solution. For example, if $b_0 = 1$ and $\delta = 0$ then $b_t = 1$ for all t and so profits are always increasing in q. This should lead to an upward distortion in p.

What can we conclude about competition with an OSS product in the baseline model? There are two effects. First, from a static perspective, competition reduces prices, but prices may be nonmonotone in the quality of the OSS. From a dynamic perspective, if the altruism parameter is not too large, a forward-looking commercial firm further reduces price in order to slow the improvement of the OSS product. The size of this distortion varies with the quality of the OSS product and the size of the altruistic programmer community. The distortions, however, should not affect the qualitative result that the OSS quality will evolve toward a positive-quality steady state from any initial condition with nonzero quality.

If the altruism parameter is very large, the intuitive result about dynamic considerations leading the commercial firm to lower its prices may not hold everywhere, and there may be states of the system (when there are many altruistic programmers, for example) that the commercial firm prices higher than in a static environment in order to induce consumers to switch away from its product in the short run.

5 Models with Customer Support

Our baseline model neglected a feature of OSS that has received a lot of attention in the descriptive literature about OSS (Shah (2004, 2006)). In particular, in many OSS projects only a small fraction of the community actually contributes to the code base. More people seem to help out by providing support service to new users, such as helping people learn to install and use the software by answering questions posted to bulletin boards. Providing this support is probably quite important for many products. Shah (2004, 2006) notes that

users providing this casual kind of support appear to have a shorter term period of active involvement with the project. That is, many consumers adopt the software, receive help from others, and then proceed to provide help to others for a period of time. In contrast, experienced programmers do not spend time answering "basic" questions for "newbies."

This type of phenomenon can have important implications for the dynamics of OSS, since it suggests that a regular flow of new users is important for maintaining customer service. The behavior of users can be understood through the lens of altruism that depreciates over time, and perhaps also due to the decline in intellectual satisfaction from answering similar questions over a long period of time. Thus, we consider a model like that of the previous section but with two populations: a unit mass of software programmers and a mass m of "users" who potentially contribute by providing service rather than new code.

Suppose that when a "user" encounters a need he or she cannot meet the need by open source unless the code has that feature *and* he or she gets help from another user. To keep the specification similar to the above model (but slightly simpler) we assume that users' needs that would be met with open source (if this is possible) arise according to a Poisson process with parameter γ_u, that users who meet their needs using open source become altruistic with probability α_u, and that their altruism decays according to a Poisson process with parameter δ_u. Assume that the probability of being able to use the code is $q_t f(mc_t)$, where c_t is the fraction of users who are altruistic at t and f is some concave function.[2]

There are several interesting options for extending our specification of software programmer preferences. We proceed to analyze a couple of these.

5.1 Code-base altruism

One way to extend the model is to assume that software programmers do not need service, and that they receive altruism benefits directly from increasing the code base (as opposed to indirectly through providing benefits to other users). This might correspond more to feelings of intellectual satisfaction or scientific achievement from contributing to a high-quality product. In this case, the q and b dynamics of the model are identical to those in the benchmark model, since users do not have an impact on programmers' objectives.

The fraction of users who are altruistic then evolves according to

$$\dot{c}_t = \alpha_u \gamma_u (1 - c_t) q_t f(mc_t) - \delta_u c_t$$

[2] One might want to specify this $f(c_t)$ to reflect that each user will only provide a fixed amount of customer service that is independent of the size of the user population.

From our analysis of the baseline model, we know that $(q_t, b_t) \to (q^*, b^*)$ from any initial condition other than $(0,0)$ provided that $\alpha\gamma_b\gamma_q - \beta\delta > 0$. When the latter condition holds, the dynamics of c for large t are then approximately

$$\begin{aligned}\dot{c}_t &= \alpha_u\gamma_u(1-c_t)q^*f(mc_t) - \delta_u c_t \\ &= \alpha_u\gamma_u(q^*f(mc_t) - \delta_u c_t) - q^*f(mc_t)c_t.\end{aligned}$$

Proposition 11 *If $\alpha_u\gamma_u q^* m f'(0) < \delta_u$ then in the limit use of the software goes to zero.*

If $\alpha_u\gamma_u q^* m f'(0) > \delta_u$ then use will not converge to zero if $c_0 > 0$. In the special case where $f(x) = x/m$, the fraction of users who are altruistic converges to $c^* = \frac{\alpha_u\gamma_u q^* - \delta_u}{\alpha_u\gamma_u q^*}$.

This result shows that service issues can lead an OSS project to be something that is tailored for programmers, but does not meet the needs of ordinary users. Industry observers have commented that OSS projects tend to be biased in this direction, and that commercial products tend to cater more to less sophisticated users. The result highlights the important role played by the slope of the "service function" f at 0 : it is important that the first few users are able to effectively support other users in order to prevent the collapse of service. Clearly, if $f'(0)$ is large (e.g. if one user is able to answer all questions for incoming users), collapse of the user base is not a concern. This suggests that when trying to get an OSS project off the ground in terms of user adoption, it may make a big difference if a few committed participants in an OSS project provide a lot of initial support.

Even when collapse of the user base is not a concern, when δ_u is large (so that user altruism depreciates quickly), service issues can greatly limit the use of the product. Again, this result is consistent with observations by industry observers that support is a critical issue for OSS projects. However, by assumption, low support does not limit the development of the project, just the rate of user adoption. We consider in the next section a perhaps more realistic variant of the model, where programmer motivation depends on the size of the user base.

5.2 User-motivated altruism

Surveys of OSS participants (e.g. Shah (2004, 2006)) indicate that programmers want to have an impact with their contributions, much as academics do. They enjoy being part of important projects, including projects that have a large user base. In particular, they are motivated by the impact of their contributions to that endeavor, and programmers tend to monitor discussions of features they have developed. This suggests that a model

should incorporate a relationship between altruism and the extent to which code is helpful to casual users.

The flow rate at which any feature will meet users's needs is $m\gamma_u f(mc_t)$. If we assume that programmer's have "myopic altruism" in the sense that the altruism benefit depends on the flow rate of use of the feature at the time of development (rather than some infinite horizon discounted measure of total use), then the altruism benefit from contributing a feature at t is $am\gamma_u f(mc_t)$.[3]

A simple way to incorporate the idea that programmers are more likely to develop a feature if the feature will be used more is to assume that B_{it} is always greater than $B_0 + E$ and that the publication cost K is a random variable distributed uniformly on $[0, a]$.[4] This implies that the probability that an programmer decides to contribute a feature to the code base is $m\gamma_u f(mc_t)$.[5]

The evolution of q_t and b_t is no longer separable from the evolution of c_t.

Proposition 12 *The dynamics of the system are given by*

$$\begin{aligned}
\dot{q}_t &= \gamma_q(1-q_t)b_t m\gamma_u f(mc_t) - \beta q_t \\
\dot{b}_t &= \alpha\gamma_b(1-b_t)q_t - \delta b_t \\
\dot{c}_t &= \alpha_u\gamma_u(1-c_t)q_t f(mc_t) - \delta_u c_t
\end{aligned}$$

As above, it is always a steady state to have no activity.

Proposition 13 *The system always has* $(0,0,0)$ *as a steady state.*

To analyze the stability of the zero activity steady state we linearize the dynamics in a neighborhood of $(0,0,0)$. Assuming that f has a finite derivative at 0 the first order approximation to the dynamics is

$$\begin{aligned}
\dot{q}_t &\approx -\beta q_t \\
\dot{b}_t &\approx \alpha\gamma_b q_t - \delta b_t \\
\dot{c}_t &\approx -\delta_u c_t
\end{aligned}$$

If we write this in matrix form as at $(\dot{q}, \dot{b}, \dot{c}) = A(q, b, c)$, then the A matrix is negative definite. This implies

[3] This also assumes as we've done implicitly throughout that the fact that others might invent the feature anyway in the future also doesn't affect altruism benefits.

[4] The assumption on B_{it} implies that the active margin is between developing versus developing and contributing. The expressions would be more complicated if lower "altruism" benefits led engineers to switch to the outside good. The assumption also implies that $\gamma_b = \gamma_q$.

[5] This assumes that the expression for the altruism benefit is always less than one.

Proposition 14 *The steady state at* $(0, 0, 0)$ *is locally asymptotically stable.*

Note that the behavior of this model is qualitatively different from the model with "code-based altruism." To have any chance of succeeding, an open-source project will need to be pushed to a sufficient level of development by some mechanism other than the altruism-fed growth of our model. This suggests an important role for highly motivated and altruistic "founding members" of an OSS project, and in particular, these members need to both develop software and provide user support.

Proposition 15 *For some parameters, the steady state at* $(0, 0, 0)$ *will be a global attractor. For other parameters the system will also have a steady state with* q*,* b*, and* c *positive.*

To see that the zero-quality steady state can be unique, note that $\dot{q}$ and $\dot{b}$ in this model are always less than they were in the baseline model. In that model, (q_t, b_t) always converged to zero if $\alpha\gamma_b\gamma_q < \beta\delta$. Hence, with that parameter restriction q and b will also converge to zero in this model.

When this happens, c must also converge to zero.

To see that there can also be steady states in which the open source software is successful note that for $\delta_c = 0$ and $c_0 = 1$ we have $c_t = 1$ for all t. The system is then just like the previous system with the substitutions $\gamma_b\prime = \gamma_b$, $\gamma_q\prime \equiv \gamma_q m\gamma_u f(m)$ and $\alpha\prime \equiv \alpha$. If we the primitives of the model are such that $\alpha\prime\gamma\prime_b\gamma\prime_q > \delta\beta$, the system will have a steady state with q^* and b^* positive.

Note that the model of this section has more nuanced predictions about what makes for a successful launch of an OSS project. For example, how high should (q, b, c) be in order to get off the ground. The example given above indicates that quality and programmer altruism can be quite low if the customer base is high and altruism among customers does not decay too much.

We can also consider how a commercial product should compete against an OSS project whose dynamics follow the "user-motivated altruism" model. There is potentially a much larger scope for strategic pricing behavior by a far-sighted commercial firm. In particular, the commercial firm may be able to price low enough to push the OSS into the basin of attraction of the zero-quality steady state. The commercial firm can consider poaching ordinary users and providing high levels of support for those users, thus diminishing the motivation of the developers of the OSS project. If the OSS project has a mix of programmers motivated by users and those motivated by the quality of the code, the commercial

firm can push the OSS towards a programmer-oriented niche, thus protecting its market with ordinary users.

6 Conclusion

In this paper, we have developed several simple models of the dynamics of OSS. We have also explored the implications of these models for (i) successful initial launches of OSS projects and (ii) competing with OSS projects.

In our base model OSS will always has a steady-state with zero activity even if it also has a steady state with positive activity. Our model, however, is not like a standard network externality model – the system converges to the higher steady state given any initial boost no matter how small. We also found that, although the dynamics of OSS projects typically vary with parameters and state variables in intuitive ways, it is possible that increasing the quality of an OSS can have perverse effects. We can also observe nonmonotone dynamics with quality or the population of committed programmers initially decreasing and then later increasing toward the steady state.

The fact that our base model does not have multiple stable equilibria may be a useful insight into OSS movements, but we think of it more as pointing out that one must incorporate other elements into a model to explain why the way in which an OSS product is launched could matter in the long run. Our analysis of user support is one such extension. It illustrates that it may be difficult to get an OSS off the ground without a core group of founders committed to providing customer support. If programmers are motivated by the size of the user base, such considerations may make it impossible to get an OSS project started, at least for some parameter values.

Commercial firms competing with OSS projects can benefit from strategic foresight. Generally, a far-sighted commercial firm should price lower than a short-sighted one, because keeping price low slows the quality growth of the OSS. Yet, following the intuition given above, this many not hold at very high levels of quality. When user support is an important phenomenon, and when user altruism depreciates over time, strategic pricing by a commercial firm can eliminate the user base of an OSS project. If a primary motivation for programmers is the size of the user base who will use additional features, strategic pricing can potentially push an OSS into a zero-quality steady state.

Many avenues remain to be explored. Our models are very stylized. An important feature absent from our model is forward-looking behavior by programmers and users. Although probably some fraction of the real-world populations of these agents would satisfy

an assumption of myopic behavior to a first approximation, we have eliminated any role for expectations about the future growth of an OSS project. Industry observers have discussed the "FUD" tactic for competing with OSS: "Fear, Uncertainty, and Doubt." When adoption of a software product is costly and there are switching costs, customers will naturally consider whether a product will be available and supported in the future. If a commercial product can affect these expectations, either through a reputation for aggressive pricing or through other means, we would also expect that the dynamics of OSS will be affected. Programmers may also consider future benefits of their contributions, and they may be more motivated to contribute to a project they expct to succeed in the future.

Another avenue for further research concerns the governance structure of OSS projects, and how that interacts with incentives for altruism. Shah (2004, 2006) provides suggestive evidence about the impact of alternative governance structures, and it should be possible to extend our model in that direction.

References

[1] Akerlof, George A. (1982): "Labor Contracts as Partial Gift Exchange," *Quarterly Journal of Economics*, 97 (4), 543-569.

[2] Casadesus-Masanell, Ramon and Pankaj Ghemawat (forthcoming), "Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows." *Management Science*.

[3] Johnson, Justin (2002): "Open Source Software: Private Provision of a Public Good," *Journal of Economics and Management Strategy*, 11 (4), 637-662.

[4] Johnson, Justin (2004): "Collaboration, Peer Review and Open Source Software," Mimeo, Cornell.

[5] Lakhani, Karim and Robert G. Wolf (2003): "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Shource Software Projects," MIT Sloan Working Paper No. 4425-03.

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

[7] Lerner, Joshua and Tirole, Jean (2005a): "The Scope of Open Source Licensing," *Journal of Law, Economics, and Organizations*, 21.

[8] Lerner, Joshua and Tirole, Jean (2005b): "The Economics of Technology Sharing: Open Source and Beyond," *Journal of Economic Perspectives* 19 (2), 99-120.

[9] Rabin, Matthew (1993): "Incorporating Fairness into Game Theory and Economics," *American Economic Review*, 83 (5), 1281-1302.

[10] Raymond, Eric S. (2001): *The Cathedral and the Bazaar*, O'Reilly.

[11] Shah, Sonali (2004): "Understanding the Nature of Participation & Coordination in Open and Gated Source Software Development Communities." *Proceedings of the Sixty-third Annual Meeting of the Academy of Management* (CD), ISSN 1543-8643.

[12] Shah, Sonali (2006): "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development." *Management Science*.

[13] Tirole, Jean (1988): *The Theory of Industrial Organization*. Cambridge MA: MIT Press.