Dynamics of Open Source Movements[1]

Susan Athey and Glenn Ellison

July 2012

[1]Stanford University and Massachusetts Institute of Technology. E-mail: athey@gsb.stanford.edu, gellison@mit.edu. We are grateful to the Toulouse Network for Information Technology and to the National Science Foundation (grants SES-0550897 and SES-0351500) for financial support. We thank participants in the 2005 Toulouse Network conference, in particular Josh Lerner and Jean Tirole, for helpful comments.

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. Programmers who have used open source software are motivated by reciprocal altruism to publish their own improvements. The evolution of the open-source project depends on the form of the altruistic benefits: in a base case the project grows to a steady-state size from any initial condition; whereas adding a need for customer support makes zero-quality a locally absorbing state. We also analyze competition by commercial firms with OSS projects. Optimal pricing policies again vary: in some cases the commercial firm will set low prices when the open-source project is small; in other cases it mostly waits until the open-source project has matured.

1 Introduction

Open source software (OSS) has many varieties, but common features are that code is freely available and contributions are made by a diffuse set of programmers often working as volunteers. Well-known success stories include Linux, Apache, which dominates the market for web servers, and PERL and PHP, which are leaders in scripting software. But OSS is a much broader phenomenon: as of July 2012, SourceForge.net hosted 324,000 OSS projects developed by 3.4 million programmers. There is a great deal of diversity in project characteristics and outcomes: projects aim to serve very different user bases; have different internal organizations; and some have thrived while others have risen and fallen (or never risen at all). There is also diversity in the relationships between commercial firms and OSS. Some software companies compete directly with open source projects in critical areas, e.g. Windows Server competes with Apache, but it is also common for firms to actively support open source projects.[1]

Though recent, the literature on open-source software has developed rapidly. It now contains both enlightening theoretical papers and convincing empirical analyses that present findings derived from diverse methodologies. The largest part of this literature focuses on why individual programmers join open-source projects and contains evidence for the importance of both economic career-concern and noneconomic intrinsic motivations.[2] A smaller literature presents insights on competition between open-source and traditional software products.[3]

[1] Lerner et al. (2006) report that the fraction of corporate contributors to open source projects ranges from 22% for the smallest projects to 44% for the largest. IBM is a particularly noted supporter of open source and claims to have invested billions. A sample description (in an interview of James Stallings by Linuxplanet.com) is:

> LP: I'd like to ask you a few things about IBM's Linux Technology Center. You have about 250 people working there. Can you tell me exactly what they're doing with their time? Are they helping customers or developing the kernel, or doing other work?
>
> Stallings: They're making contributions. Their full time job is making contributions to the kernel. That's it. They don't have another job sweeping the floor or working on Websphere or anything like that.

[2] See, among many others Lerner and Tirole (2002) and Johnson (2002) for theoretical analyses, Fershtman and Gandal (2007) for empirical evidence on extrinsic motivations andd Lakhani and Wolf (2003) for survey evidence

[3] Two noteworthy papers here are Casadesus-Masanell and Ghemawat (2006) and Economides and Kat-

Our paper departs from most of the literature in a few ways. First, we conceptualize an open source software project as a community of programmers within which reciprocally altruistic preferences are a driving force. Second, our analysis differs in our focus on the dynamic life cycle as projects take off, grow, and/or decline. Third, some of the specific questions we examine differ: why programmers go to the effort to commit code to the project; how projects are affected by competition with for-profit firms; and how development is affected by the importance of customer support.

Our formal model assumes that at each point in time the open-source product can meet a fraction of the "needs" of a community. We refer to the fraction of needs that a product meets as its "quality" and assume that quality naturally depreciates over time (perhaps because new needs such as compatibility with new hardware and software become relevant). Quality increases if these declines are more than offset by improvements made by a community of programmers. We assume that programmers are motivated both by their own needs and by a form of reciprocal altruism akin to that in Akerlof (1982) and Rabin (1993): altruistic feelings are activated when a programmer benefits from using the product (which is more likely when the quality is high), and then disappear at some random future time. The model is therefore at its core a dynamic system with two important state variables: the current quality of the software product; and the size of the set of programmers who currently have altruistic feelings toward the community.

Our choice to focus on reciprocal altruism is influenced by several descriptive papers. Shah (2006), for example, reports that individuals typically first become involved with an open source community when they have a need that they meet by using the software. She reports that the most commonly cited reason for remaining involved for at least a brief period is a sense of reciprocity, e.g. "Others helped me, so I should help them," and that participation becomes a hobby for a a much smaller number of deeply involved participants. The relevance of such altruistic preferences is also supported by surveys which note that programmers cite such motivations along with other intrinsic (enjoyment of intellectual

samakas (2006).

stimulation) and extrinsic (career concerns) motivations.[4] We do not wish to deny the importance of these other factors. Instead, as in Johnson (2002) our view is that we have incorporated them albeit crudely in the form of the fixed net benefits and effort costs that programmers are assumed to accrue when they make use of the software and develop and contribute improvements.

Our initial 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 features and become altruistic are large enough relative to the rates at which software features and programmer altruism depreciate, 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." In our baseline model, starting from any initial condition with some features implemented in the software, the system converges to this the higher steady state. Hence, initial conditions and founder behavior have little to do with the long-run success of the project. They only affect the path that the project takes along the way to this steady state.

We turn next to consider competition between a commercial software firm and an OSS project. 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. In the most natural case, the commercial firm strategically prices below its static best response in an effort to slow the growth of the OSS project. The magnitudes of the strategic price distortions depend on several factors including the loss in short-run profits from a price cut; the degree to which a price cut affects the state variables; and the persistence of changes in the state variables. We present some numerical examples, and note that strategic incentive to cut prices can be largest when the OSS project is near its steady-state size, because in other states evolutionary forces are more powerful and changes to the state variables are less persistent.

[4]See Ghosh et al. (2002), Hertel et al. (2003), and Lakhani and Wolf (2005).

Section 5 extends the model to incorporate another aspect of open-source communities that features prominently in some descriptive papers: community members who lack the expertise to contribute code, but are actively involved in providing "customer support" to new users such as answering easy questions that expert programmers have neither the time nor the inclination to deal with. The extended model adds a user population and the fraction of this population currently feeling altruistic becomes a third state variable.[5] The literature on public goods has noted that outcomes in some models depend on the form of altruism: action-oriented altruism in which agents receive "warm glow" utility from the actions they take in the attempt to help others; and "pure" or outcome-oriented altruism in which agents only care about the benefits others receive.[6] A related modeling choice arises in our model: agents could have "code-based" altruism in which they receive utility from contributing to the open source project; or they could have "user-motivated" altruism in which they value making contributions more when more users will take advantage of them. One result of this section is that the system dynamics are qualititatively different depending on the form of programmers' altruism. Another is that the model can have "critical mass" effects: with user-motivated altruism the model can be such that a substantial initial push is needed to prevent the project from collapsing to a zero-quality state. (This in turn can qualitatively affect the behavior of competitors.)

As noted above, our paper is contributing to a literture on open source that is now substantial. Two benchmark models are Lerner and Tirole (2002) and Johnson (2002). The former focuses on the economics of open source contributions. 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. Johnson (2002) models open source software as a public good contribution problem. This provides a framework for welfare comparisons and brings out issues like

[5]Kuan (2001) examines the problem of fixing bugs which is another way in which low-skilled users may contribute.

[6]See, for example, Andreoni (1989), Rose-Ackerman (1996), and Francois and Vlasopoulos (2008).

free-riding, duplication of effort, and the potential for mismatch between developer effort and user preferences. Other noteworthy papers on user motivations and their implications for open source organization include Lerner and Tirole (2005a) and Johnson and Myatt (2006).

Our focus on altruistic preferences is motivated by the survey literature including Ghosh et al. (2002), Herter et al. (2003), Shah (2004), and Lakhani and Wolf (2005). Lakhani and von Hippel (2003) and Shah (2006) highlight the importance of user support.

There is also a prior literature addressing questions related to those in our analysis of competition between commercial and open source products. Schmidt and Schnitzer (2003) discuss welfare issues in competition between open source and commercial software. Mustonen (2003) models both product-market competition and competition for programmers. Economides and Katsamakas (2006) study a platform-competition problem in which a critical consideration is the variety of complementary applications that will be developed on each platform. Casadesus-Masanell and Ghemawat (2006) analyze a dynamic competition model in which consumer demand has network externalities and commercial firms are forward-looking in their pricing. Our approach is complementary in that we allow for much richer dynamics in the OSS product and 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 quality and can be thought of as an endogenous network externality.

Our paper is also broadly related to the recent literature on boundely rational industrial organization.[7] Whereas most of this literature is concerned with how rational firms price when confronted with behavioral consumers, our paper can be thought of as more closely related to the older literatures discussed in Ellison (2006) in which the firms were non-rational entities.

[7]See among others Della Vigna and Malmendier (2004), Ellison (2005), Gabaix and Laibson (2006), Spiegler (2006a, 2006b), Eliaz and Spiegler (2008), Heidhues and Koszegi (2008), Kamenica (2008), and Ellison and Ellison (2009).

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 with 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 are myopic utility maximizers. 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 $\text{Prob}\{a_{it} = 0 | a_{it-dt} = a\} = \delta dt$ and $\text{Prob}\{a_{it} = a | a_{it-dt} = 0\} = 0$.

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.

Note that although we have not explicitly included any career-concerns or intrinsic-enjoyment motivation for contributing to the open source project, one can think of such motivations as being reflected in the exogenous costs E and K of programming and contributing code to the project. For example, K can be thought of as the net cost of contributing code, which is the difference between the effort costs incurred in going through the submission process and any enjoyment or career concerns benefit that accrue.[8]

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 their payoff from the current action.[9]

3 Developer Behavior and Community Dynamics

In this section we analyze the model described in Section 2. Our main result is a characterization of the dynamic system showing that two qualitatively different behaviors are possible: for some parameters the project is doomed to fail; for others there is both a zero-quality-zero-participation steady state and a steady state where the project achieves

[8] The empirical evidence in Fershtman and Gandal (2007) and other papers and the simple fact that open source projects establish hierarchies and are careful to credit contributors suggest that such motivations are important. The degree to which our model can capture them is limited by our assumptions that K is positive and that K does not vary with the size of the community, the quality of the project, etc.

[9] The myopic maximization can reflect some long-run thinking, e.g. career concerns benefits, by including the net present value of the long-run benefit in the instantaneous payoff that programmers receive upon contributing. Given that agents only observe aggregates when making their decision, the assumption of myopic play is similar to assuming that players maximize their lifetime discounted utility given the specified payoffs. 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.

a degree of long run success. We derive expressions for participation levels and software quality in the successful steady state and describe the dynamics of the system.

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. First, we note that it is only when altruism is sufficiently strong that open source projects can succeed in our model.

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

Next, we present a few results on programmer behavior. What programmers do when open source software can meet their needs is an immediate consequence of our assumptions.

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

Meeting the need by programming is dominated by using open source because $E > 0$. The other 'program-and-contribute' option is assumed to only be available if the feature is not already in the open source package.[10] When the open source project cannot meet the need programmer behavior is a little more complicated.

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

[10] Note that we are assuming away the coordination problem discussed in Johnson (2002) which would arise if programmers were unaware of whether others were simultaneoulsly working on the same improvement.

only for their own private benefits, and they will never share their code after developing it.[11] 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 will assume that altruism is sufficiently strong so that altruistic programmers will contribute any improvements they have made:

Assumption 1 *Assume $a > K$.*

It then follows immediately that:

Corollary 1 *Under Assumption 1 the equilibrium strategies are*

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

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 (assuming it exists) 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

[11] In practice there could be some private benefits from publishing code either of the career-concerns variety or because publishing code leads others to support or improve it. Our assumption that $K > 0$ is an assumption that any such benefits are less than the costs of submitting code to the project.

programmer is confronted with a need that (assuming it is not already in the OSS) 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 quality q_t and of the mass of programmers b_t are*

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

We now provide a series of results to characterize the behavior of this dynamic system. Roughly, the results say that the dynamic system will exhibit one of two behaviors depending on the parameter values. One possibility is that the project is doomed: there could be a temporary improvement on one dimension from some initial conditions but the project eventually decays to a zero quality-zero altruism limit. The second more interesting possibility is that there long-run success is possible. In this case, there is a unique steady state quality-altruism pair that the system evolves to from almost any initial condition (a zero quality, zero altruism steady state exists but is unstable.)

An outline of the argument is that we begin by deriving two relevant threshold functions: the fraction of the programmer population who must be altruistic in order to replace depreciating features and thereby sustain quality at its current level; and the minimum quality level that the software would have to have in order to attract enough programmers to sustain the the altruistic population. The relative positions of these two curves turns out to be crucial to the nature of the dynamics. Intuitively, software quality and the size of the altruistic population can grown in tandem only if there is a growth path that remains above both curves.

We begin by describing the $\dot{q} = 0$ curve which gives the minimum size b of the altruistic population needed to maintain quality at any given quality level q. It is defined only for some values of q because when $q > \gamma_q/(\beta + \gamma_q)$ quality depreciation is unavoidable: we have $\dot{q} < 0$ even if all programmers are altruistic. 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

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

The solution to his equation is

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

Thus, the function is proportional to the ratio of the rate of decay of quality to the arrival rate of needs that would lead an altruistic programmer to develop and contribute (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 size of the altruistic population needed to sustain quality at q, is convex and strictly increasing in q on $(0, \gamma_q/(\beta+\gamma_q))$. It satisfies $b_{\dot{q}}(0) = 0$, $b_{\dot{q}}(\gamma_q/(\beta+\gamma_q)) = 1$, and $b'_{\dot{q}}(q) = \beta/\gamma_q(1-q)^2$.*

Intuitively $b_{\dot{q}}(q)$ is increasing when more altruistic programmers are required to sustain software quality at a higher level. This follows immediately from our assumptions that quality depreciates proportionally, and that new code is only published if a programmer develops the code and feels enough altruism to outweigh the publication costs. The convexity results from a second mechanism acting on top of the the proportional depreciation: when quality is high fewer new features are being developed because the software already meets many needs.[12]

Now we turn to consider the $\dot{b} = 0$ curve – the relationship that describes how high quality must be in order to attract newly altruistic programmers at a rate sufficient to sustain the size of the altruistic population at any given size b. For the purposes of the analysis, however, we will also characterize this as a function of q, i.e. we ask how large of

[12]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.

an altruistic population can be sustained if the quality level is q. 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

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

Solving this equation we find

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

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 $b'_{\dot{b}}(q) = \delta\alpha\gamma_b/(\alpha\gamma_b q + \delta)^2$.*

Intuitively, $b_{\dot{b}}(q)$ is increasing if for higher values of quality a larger altruistic population can be sustained. This holds because the number of programmers who benefit from the software is larger when q is high, this leads to a larger inflow of newly altruistic population, which can offset the proportional depreciation of altruism in a larger altruistic population. The function $b_{\dot{b}}(q)$ 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 the $\dot{b} = 0$ curve. Similarly, the one can think of q as evolving in the direction of the $\dot{q} = 0$ curve.

The system is in steady state where $b_{\dot{q}}(\cdot)$ and $b_{\dot{b}}(\cdot)$ intersect: quality is attracting new altruistic programmers at the exact rate at which altruistic programmers are leaving; and the altruistic programmer population is developing new features at a rate that exactly offsets the quality depreciation. 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.

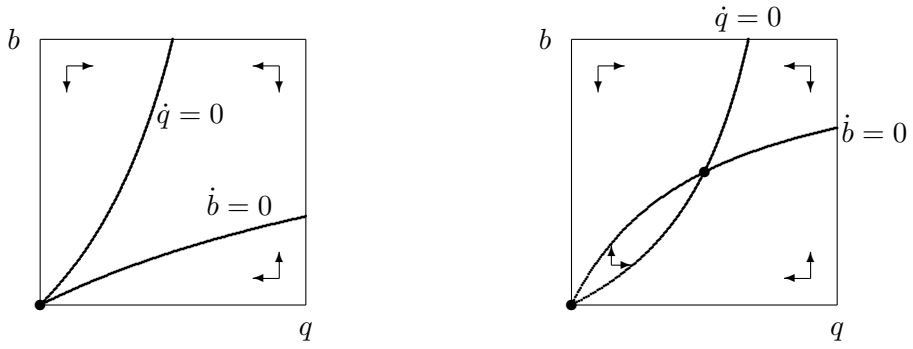

Figure 1: Model Dynamics: the left panel has $p = 0.5$ and $\gamma_b = \gamma_q = \beta = \delta = 1$. The right panel has $p = \gamma_b = \gamma_q = 1$ and $\beta = \delta = 0.5$.

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

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

This brings us to the main result of this section: the dynamic system may be of one of two qualitatively different forms depending on the parameters of the model.

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 frequency with which programmers encounter needs that can be met using a complete open-source product, the consequently triggered feelings of altruism, and the arrival of situations where altruistic programmers would contribute new features must be sufficiently large relative to the speed at which altruism and features depreciate. Note that the depreciation of altruism and the depreciation of features enter symmetrically.

In addition to knowing whether success is possible it is interesting to ask the quantitative question of *how* successful the project will be in the successful steady state. 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 features 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 find that the steady state fraction of altruistic programmers is 1 ($b^* \approx 1$); and 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 $(q, b) = (0, 0)$ 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 a few committed (altruistic) programmers will eventually reach a steady state that is predetermined given parameters. This contrasts with some informal characterizations of open source projects as needing some kind initial "big push" to have a chance of success. This suggests that empirical analyses of whether a big push is necessary would be interesting, and that it would be interesting to investigate modifications that could be made to our model to create big-push dynamics if that is the empirical reality.[13]

[13] We present one model that has this feature in Section 5.

It can also be shown that projects tend to grow with quality and fraction of programmers who are altruistic 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

Many software markets include both open source and commercial products. Microsoft's Windows Server competes with open Apache (and Nginx) in the web server market. MySQL was seen by some to be an important competitor to Oracle in relational databases. The open source Firefox browser competes with Microsoft's Internet Explorer, Apple's Safari, and Google's Chrome. An important question for both public and business policy concerns how competition between an OSS product and a commerical product differs from competition between two commercial products or from monopoly pricing. Commercial firms also sometimes support open source projects, although we will focus here on the competitive interaction.

We model a single commercial software product competing with a single OSS project. We 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 omits direct network effects of the form analyzed by Casadesus-Masanell and Ghemawat (2006). Instead, we consider the "network effects" that arise endogenously in our model through the provision of features by altruistic programmers, which has the

effect of making users derive more utility from the open source product if more have used it in the past. Indeed, a primary focus will be on how the commercial firm's price will reflect the desire to limit these network effects in both the short and long run.

We begin by deriving demands. 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 firm 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

$$\pi(p; q, b) = p\left(qG(v-p) + (1-q)(1-b)G(v-p+E) + (1-q)bG(v-p+E-(a-K))\right).$$

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 case whre the commercial firm is worse off when OSS quality is lower seems unlikely to be relevant in practice – in our model it requires that programmers' altruistic motivations be very strong and our model also omits customers who are not programmers and hence clearly more likely to purchase commercial software when OSS quality is lower. But for completeness we include the case in this and some other results.

4.1 Static profit maximization

In this section we consider the commercial firm's static monopoly pricing problem both in general and in a tractable special case. The static profit maximization problem is

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

Note that it is of the form

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

with $d_1 + d_2 + d_3 = 1$, where d_j is the fraction of total firm consumers coming from $j \in \{1,2,3\}$, corresponding to the three groups of consumers described above, and $\hat{v}_j$ is the net benefit to the consumer of type j 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

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

where $Q(p) \equiv \sum_j d_j G(\hat{v}_j - p)$ is the total quantity that the commercial firm sells at price 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

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

Firms charge a higher price when the effort costs E are higher and a lower price when the altruistic motivation a is stronger. 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:

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

In the standard case ($a - K < E$) where the commercial firm is better off when OSS quality is lower we obtain the intuitive result that a higher quality OSS product leads to a lower price for the commercial firm. The commercial firm's price is less sensitive to open source quality when more programmers are altruistic. It is also straightforward to show that the price is lower when more programmers are altruistic. In the extreme case mentioned earlier where the commercial firm is better off when OSS quality is high, the commercial firm can increase its price in response to a higher quality competitor.

4.2 Dynamic profit maximization

Our primary interest in this section is in how commerical firms strategically "distort" prices away from the static optimum in order to affect the growth of open-source competitors. Here we formulate the dynamic profit-maximization problem that we will analyze to address this question. Specifically we consider the following dynamic problem:

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

subject to

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

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 we have not made OSS programmers forward-looking in this model. Instead, we continue to assume that they make myopic choices based on the flow benefits of programming, publishing, or using the commercial product.[14]

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.

The dynamic price distortions are less straightforward when $a - K > E$. Recall that in this case profits are increasing in q when b is large because programmers are sufficiently altruistic so as to make them more likely to choose OSS when it works less well (because they gain utility from improving it). Choosing a higher p increases $\dot{q}$ (although it also increases $\dot{b}$) so offsetting effects would need to be considered. This case could have interesting dynamics to explore, but does not seem likely to be empirically relevant, so we will not focus on it in the remainder of this paper.

4.3 Magnitudes of stratgic price distortions

Consider the "standard" case where a commercial firm does better when the open source product is lower in quality ($a - K < E$). Lowering prices away from the static optimum has no first-order cost and gives a first-order dynamic benefit, so the commercial firm will "distort" prices downward from the static optimum. The magnitude of the difference between static and dynamically optimal prices will depend on several fractors: there is less cost to distorting prices when the commercial firm's quantity is low; the benefit from distorting prices by a given amount is larger when the effect on the state variables (q, b) is larger; and the benefit of shifting the state variables by a given amount is larger when the dynamics are such that the shifts will be more long-lived.

To get some feel for how these considerations play out, Figure 2 graphs the difference between the static optimal and the dynamic optimal prices as a function of q and b for one

[14] Atomistic programmers would have no dynamic incentives other than that mentioned earlier – by becoming altruistic a programmer affects future altruism utility – but strategic behavior by a forward-looking leadership team could be an interesting topic for future research.

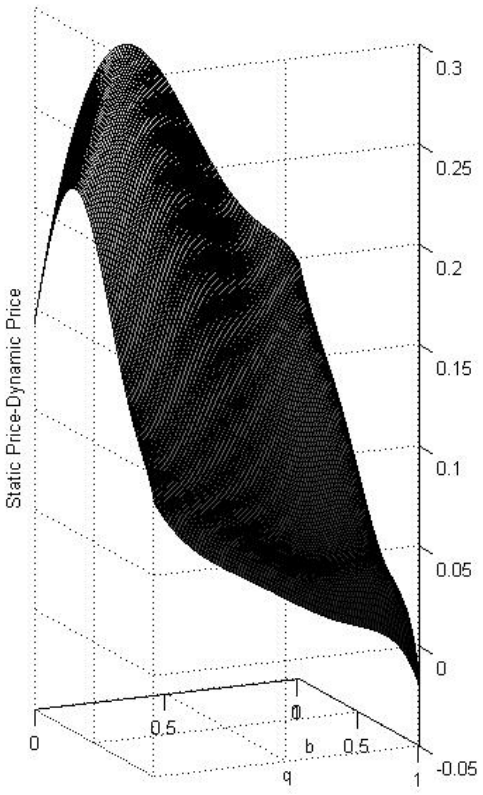

Figure 2: Strategic price distortions: the difference between the static and dynamic optimal price as a function of (q, b) for one set of parameters.

set of parameters.[15] In this case, strategic price distortion is largest when the system is near the steady state of the static model ($(q, b) \approx (0.3174, 0.2876)$) and drops sharply near each of the extreme states, including that where the open-source product is very weak.

Most of the intuition for this is obtained from thinking about the vector field describing the evolution of the system under static-optimal pricing, which is graphed in Figure 3. The incentive to distort prices is high when the state is near the steady state, because the system moves very slowly in these cases and hence manipulations that shift the state pay off for a long period of time. Distortions are very small when the state is close to $(1, 0)$, $(0, 1)$, or $(1, 1)$ for analogous reasons: the dynamics move away from these points very quickly so the benefits of manipulation are small. The reason for not pricing aggressively when the open-source product is in its infancy are different. The system moves slowly in a neighborhood of $(0, 0)$ so shifts in the state yield long-lasting benefits. But price cuts have only a small effect on the evolution of the system ($\frac{d\dot{q}}{dp}$ and $\frac{d\dot{b}}{dp}$ are both zero at $(q, b) = (0, 0)$), so there is not much incentive to sacrifice short-run profits for this reason.

Because the commercial firm mostly distorts prices when prices are near the steady state for these parameter values, the dynamics are qualitatively similar regardless of whether the monopolist practices static- or dynamic-optimal pricing. The steady state size of the open source project, however, is somewhat lower with dynamic pricing.

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: only a small fraction of the OSS community actually contributes to the code base.[16] More people help out by providing support service to new users, 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 support appear to have a shorter period of active involvement with the project. That is,

[15] Values B_{it} are assumed to be drawn from a uniform distribution on $[0, 10]$. Costs and benefits of adding features are $a = 2$, $K = 1$, and $E = 2$. Probability of becoming altruistic is $\alpha = 0.7$. Depreciation rates are $\beta = \delta = 0.5$. Other parameters are $r = 0.05$, $v = 3$, and $\lambda = 1$.

[16] Again, Shah (2004, 2006) does a very nice job of providing descriptions and analysis.

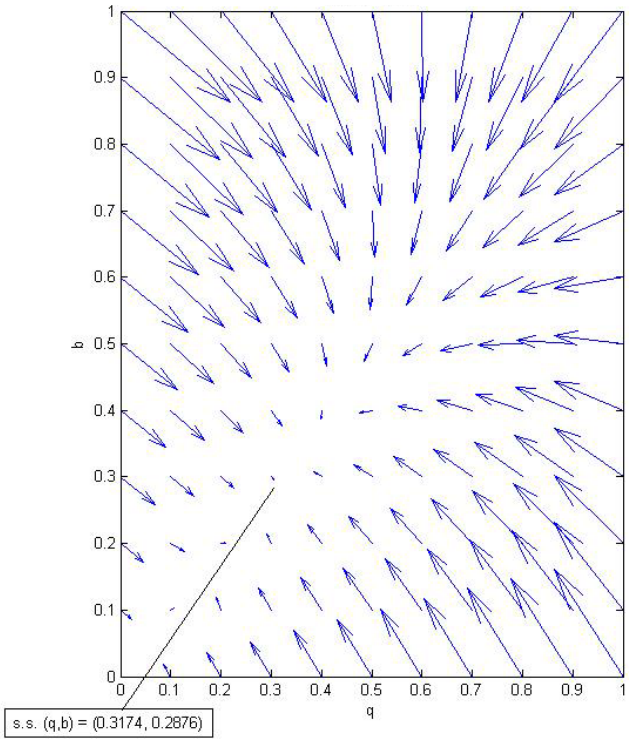

Figure 3: The vector field $(\dot{q}, \dot{b})$ of the system under static-optimal pricing.

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 rarely 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 m is the mass of users, c_t is the fraction of users who are altruistic at t and f is some concave function with $f(0) = 0$.[17]

The results of this section turn out to highlight that the impact of the need for customer services hinges on how one specifies altruism. We consider two separate formulations of programmer altruism each of which can be motivated by some findings in the literature on altruism.

[17] One might alternately want to specify this $f(c_t)$ to model situations where users provide one-on-one assistance to other users. The $f(mc_t)$ formulation in motivated by activities like compiling lists of answers to FAQ's in which a larger user base leads to a larger set of people who can produce support materials without putting extra demands on those providing the support.

5.1 Code-base altruism

One formulation of altruism that has received attention in the public goods literature is that of "impure" or "warm-glow" altruism (Andreoni 1989). Warm glow altruism is an inward-looking formulation in which atruists are assumed to derive benefits from the act of giving (as opposed to from the benefits that others derive from their gifts).[18] One way to model programmer altruism along these lines is what we call "code-base" altruism. In this formulation altruistic programmers (as before) derive utility simply from contributing to the code base. This could also capture feelings of intellectual satisfaction or scientific achievement from contributing to a high-quality product that Lakhani and Wolf (2005) find to be important.

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. It then remains only to specify how the fraction of users who are altruistic evolves. Following the discussion above we specify this as

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

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

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

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

[18] See Andreoni (1993) for some related experimental evidence.

have commented that OSS projects tend to be biased in this direction, and that commercial products cater more to unsophisticated 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

The more traditional formulation of altruism in the economics literature is the "pure" altruism model in which an altruistic agent's utility is a function of his and others' well being. Arrow (1972) also notes a potential middle ground between this and the warm glow model: "welfare is derived not merely from an increase in someone else's satisfaction but from the fact that the individual himself has contributed to that satisfaction." Surveys of OSS participants indicate that programmers want to have an impact with their contributions, much as academics do. They appear to enjoy being part of important projects, including projects that have a large user base.[19] This suggests that a model should incorporate a relationship between altruism and the extent to which code is helpful to casual users. In this section we develop a model along these lines which we term "user-motivated altruism"

[19] Shah (2006) quotes one programmer on this: "Why work on something that no on will use? There's no satisfaction there." Other supporting evidence includes that some programmers report that they monitor discussions of features they have developed even though they rarely take part in them.

and note that it leads to important qualitative changes in the dynamics.

The flow rate at which any feature will meet users's needs is $m\gamma_u f(mc_t)$. We assume that programmers' altruism benefits depend on the flow rate of use of the feature at the time of development.[20] Specifically, we assume that the altruism benefit from contributing a feature at t is $am\gamma_u f(mc_t)$.

A simple way to formulate a tractable model in which programmers will be 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 $[0, a]$.[21] This implies that the probability that an programmer decides to contribute a feature to the code base is $m\gamma_u f(mc_t)$.[22]

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* $(q, b, c) = (0, 0, 0)$ *as a steady state.*

The presence of a steady state at zero activity is not a difference from the previous model, but the nature of the dynamics in the neighborhood of this steady state turns out to be an important difference. 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

[20] One could alternately assume that altruistic benefits are some infinite horizon discounted measure of total use, but this would make the model less tractable. We also assume as we have done implicitly throughout that the fact that others might eventually have invented the feature in the future also does not affect altruism benefits.

[21] 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 $\gamma_b = \gamma_q$.

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

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

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

Note that the behavior of this model is qualitatively different from the model with code-based altruism. In our model of user-motivated altruism, an open-source project will need to be pushed to a sufficient level of development by some mechanism other than the ordinary altruism-fed growth in order to have any chance of succeeding. 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 $(q, b, c) = (0, 0, 0)$ 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 assume 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, i.e. how high (q, b, c) must be for the project to 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.

5.2.1 Competing with open source

The strategy for competing with an open-source product can be very different in our "user-motivated altruism" model. If the model has multiple stable steady states, then there is large permanent benefit from shifting the state into the basin of attraction of the zero-quality state. Hence, one would expect a commercial firm to follow such a strategy whenever the initial state is not too far from this basin.

How exactly this will be done can vary depending on the parameters of the model and whether the commercial firm has additional instruments other than price. For example, whenever $f(0) = 0$ the dynamics converge to the zero-quality from any state with $c_t = 0$. Hence, one strategy for eliminating the OSS competitor may be to take actions to attract as many ordinary users as possible, for example providing high levels of support, which will diminish the motivation of potential OSS developers. In general, an important consideration will be how far the initial state is from the basin of attraction of the zero-quality equilibrium in each dimension.

5.3 Supporting open source

Although we have focused our discussion on commercial firms competing with open source products, many commercial firms also provide a great deal of support to open source projects. The phenomenon is much broader than the well known examples of IBM's support for open source and Red Hat's building a business around Linux support: Lerner et al. (2006) attribute 44% of the contributions to the largest open source projects to corporate employees. One obvious reason for such support is that some firms profit from selling products and services that are complementary to OSS products and Mustonen (2005) notes that

support can be optimal even when products are otherwise substitutes if there are shared network-externality benefits.[23] Firms selling complementary goods could in principle support OSS projects in multiple ways: they could contribute directly to programming; and they could reduce the prices of the complements they sell to make OSS more attractive.[24]

How exactly firms would support open source would as above depend on which version of the model we examine, the model parameters, and the initial conditions. The most salient incentive is that which arises in our user-motivated altruism model when the initial conditions are in a neighborhood of the $(0, 0, 0)$ steady state. Here, a firm selling complements could potentially receive a substantial long-run benefit while incurring only a short run cost by providing programmers (or lowering prices of complements) for a limited period to push the project to the point where it becomes self-sustaining. When OSS projects are established and near a positive steady-state firms selling complements will have incentives similar to (but opposing) the competitors' incentives noted earlier: there will be an incentive to support the OSS project when it is near the steady-state quality level because the slow dynamics make the effect of such support more long-lasting. The model also points out that in some situations the form that support takes can be important. For example, in the situation noted in our analysis of the code-based altruism case where the OSS project is successful in terms of "quality" but does not attract a substantial user base because of a collapse of customer support, a commercial firm could potentially have a substantial impact by providing customer support.

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.

[23] Other reasons can also be given including that allowing employees to work on open source projects can improve their human capital and that it can be a perk which allows firms to attract employees at a lower wage as in Stern (2004).

[24] Other possibilities exist as well. For example, Oracle purchased MySQL in 2010 and Google provides most of Firefox's revenues.

In our base model OSS will always have 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.

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. This never eliminates OSS competition in our base model, but keeping price low does slow the quality growth of the OSS. How far prices are distorted depends on the parameters of the model and the current state of the OSS product. We noted that distortions may be largest when the OSS project is near its steady-state quality, because the benefits of reducing OSS quality are larger when the reductions are longer-lasting.

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.

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. There is also greater scope for commercial firms to support OSS projects in such

case as providing short-run support may change the long-run outcome in the market.

Many avenues remain to be explored. Our models are very stylized. We hope this may be an advantage in two ways: one could develop microfoundations for some assumptions to make them less sylized; or one could leave the model as it is and take advantage of the tractability to add other considerations. Some of the most important future direction may involve enriching our current specification of the costs and benefits of contributing to open source. Career concerns benefits can be seen as crudely incorporated in our current model as part of the net cost of contributing, but a richer model might incorporate interesting state-dependent variation, e.g. benefits could be larger when projects are young and growing because it is easier to advance in the project hierarchy and more future programmers will see the contribution. On the other hand, commercial firms could be more supportive of employees' contributing to established projects because the benefits to the firm from selling complementary goods and from having employees with expertise in the OSS software is likely to be larger.

One aspect of OSS communities that strikes us as potentially interesting is the heterogeneity in the governance structures of OSS projects. It seems natural that different structures could affect the rate at which programmers develop altruistic feelings. And we have left out any model of the review process that programmers must go through to get submissions of code accepted and the hierarchical structures around these reviews. Whether one wants acceptance rates to be high or low, etc. could depend in interesting ways on the form that altruism takes and could differ at different stages of the project lifecycle.

Finally, we have said little about the welfare effects of open source projects either in isolation or in competition with commercial software. Given that some European governments have adopted pro-OSS policies it is policy relevant as well as intellectually interesting to comment on whether supporting OSS projects (or even having them at all) will improve social welfare. Johnson (2002) provides some intersting observations about welfare in a public-goods model of open source, but there would be additional effects in a model like ours. Results may also depend on how one counts altruistic benefits in the welfare function.

References

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

[2] Andreoni, James (1989): "Giving with Impure Altruism: Applications to Charity and Ricardian Equivalence," *Journal of Political Economy*, 97, 1447-1458.

[3] Andreoni, James (1993): 'An Experimental Test of the Public-Goods-Crowding-Out Hypothesis," *American Economic Review*, 83 (5), 1317-1327.

[4] Arrow, Kenneth J. (1972): "Gifts and Exchanges," *Philosophy and Public Affairs*, 1 (4), 343-362.

[5] Casadesus-Masanell, Ramon and Pankaj Ghemawat (2006): "Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows." *Management Science*, 52 (7), 1072-1084.

[6] Della Vigna, Stefano and Ulrike Malmendier (2004): "Contract Design and Self-Control: Theory and Evidence," *Quarterly Journal of Economics*, 119, 353-402.

[7] Economides, Nicholas and Evangelos Katsamakas (2006): "Two-Sided Competitions of Proprietary vs. Open Source Technology Platforms and Implications for the Software Industry," *Management Science*, 52 (7), 1057-1071.

[8] Eliaz, Kfir and Ran Spiegler (2008): "Consideration Sets and Competitive Marketing," mimeo, Brown University and London School of Economics.

[9] Ellison, Glenn (2005): "A Model of Add-on Pricing," *Quarterly Journal of Economics*, 120, 585-637.

[10] Ellison, Glenn (2006): "Bounded Rationality in Industrial Organization," in Richard Blundell, Whitney Newey, and Torsten Persson (eds.) *Advances in Economics and Econometrics: Theory and Applications*, Ninth World Congress, Cambridge University Press, Cambridge.

[11] Ellison, Glenn and Sara Fisher Ellison (2009): "Search, Obfuscation, and Price Elasticities on the Internet," *Econometrica*, 77, 427-452.

[12] Fershtman, Chaim and Neil Gandal (2007): "Open Source Software: Motivation and Restrictive Licensing," *International Economics and Economic Policy*, 4 (2), 209-225.

[13] Francois, Patrick and Michael Vlassopoulos (2008): "Pro-social Motivation and the Delivery of Social Services," *CESifo Economic Studies*, 54, 22-54.

[14] Gabaix, Xavier and David Laibson (2006): "Shrouded Attributes, Consumer Myopia, and Information Suppression in Competitive Markets," *Quarterly Journal of Economics*, 121, 505-540.

[15] Ghosh, Rishab, Ruediger Glott, Bernhard Krieger, and Grigorio Robles (2002): "Free/Libre and Open Source Software: Survey and Study, Part 4: Survey of Developers," mimeo, International Institute of Infonomics, Maastricht.

[16] Heidhues, Paul and Botond Koszegi (2008): "Competition and Price Variation when Consumers are Loss Averse," *American Economic Review*, 98 (4), 1245-1268.

[17] Hertel, Guido, Sven Niedner and Stefanie Hermann (2003): "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel," *Research Policy*, 32, 1159-1177.

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

[19] Johnson, Justin (2006): "Collaboration, Peer Review and Open Source Software," *Information Economics and Policy*, 18 (4), 477-497.

[20] Kamenica, Emir (2008): "Contextual Inference in Markets: On the Informational Content of Prouct Lines," *American Economic Review*, 98 (5), 2127-2149.

[21] Kuan, Jennifer W. (2001): "Open Source Software as Consumer Integration into Production," mimeo, SSRN.

[22] Lakhani, Karim and Eric von Hippel (2003): "How Open Source Software Works: "Free" User-to-User Assistance," *Research Policy*, 32 (6), 923-943.

[23] Lakhani, Karim and Robert G. Wolf (2005): "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Shource Software Projects," in J. Feller, B. Fitzgerald, S. Hissam, and K.R. Lakhani (eds.) *Perspectives on Free and Open Source Software*, Cambridge: MIT Press.

[24] Lerner, Joshua, Parag Pathak and Jean Tirole (2006): "The Dynamics of Open Source Contributors," *American Economic Review Paper and Proceedings*, 96 (2), 114-118.

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

[26] Lerner, Joshua and Tirole, Jean (2005a): "The Scope of Open Source Licensing," *Journal of Law, Economics, and Organizations*, 21, 20-56.

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

[28] Mustonen, Mikko (2003): "Copyleft–the Economics of Linux and Other Open Source Software," *Information Economics and Policy*, 15 (1), 99-121.

[29] Mustonen, Mikko (2005): "When Does a Firm Support Substitute Open Source Programming?" *Journal of Economics and Management Strategy*, 14 (1), 121-139.

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

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

[32] Rose-Ackerman, Susan (1996): "Altruism, Nonprofits, and Economic Theory," *Journal of Economic Literature*, 34, 701-728.

[33] Schmidt, Klaus and Monica Schnitzer (2003): "Subsidies for Open Source? Some Economic Policy Issues of the Software Market," *Harvard Journal of Law & Technology* , 16 (2), 473-505.

[34] 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.

[35] Shah, Sonali (2006): "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development." *Management Science*, 52 (7), 1000-1014.

[36] Spiegler, Ran (2006a): "The Market for Quacks," *Review of Economic Studies*, 73 (4), 1113-1131.

[37] Spiegler, Ran (2006b): "Competition over Agents with Boundedly Rational Expectations," *Theoretical Economics*, 1 (2), 207-231.

[38] Stern, Scott (2004): "Do Scientists Pay To Be Scientists?," *Management Science*, 50 (6), 835-853.

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