

Dynamics of Open Source Movements¹

Susan Athey and Glenn Ellison

September 2010

¹Harvard University and Massachusetts Institute of Technology. E-mail: athey@fas.harvard.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 February, 2009, SourceForge.net hosted 230,000 OSS projects and had two million million registered users. There is a great deal of diversity in both project characteristics and outcomes: projects aim to serve very different user bases; have different internal organization; 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

Our paper is somewhat different from most of the literature. One difference is our focus on dynamics: we seek to understand patterns of growth and decline and how the composition of an OSS community changes over time. Another is the nature of our models:

¹Lerner 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.

²See, among many others we'll discuss later, Lerner and Tirole (2002) for an early theoretical analysis and Fershtman and Gandal (2007) and Lakhani and Wolf (2003) for empirical evidence of extrinsic and intrinsic motivations, respectively.

³Two noteworthy papers here are Casadesus-Masanell and Ghemawat (2006) and Economides and Katsamakas (2006).

we start from the premise that open-source contributors have altruistic motivations and develop fairly simple evolutionary models to explore how this affects the performance of the organization. Our first main focus here is on how characteristics of product, characteristics of the user/programmer population, and particularly the nature of programmers' altruistic motivations affect an open-source movement over time. We also explore how commercial software firms will price strategically when facing open source competition and what impact this has. Among our observations are that whether open-source projects face a "critical mass" hurdle depends on the form of altruistic preferences; and that competitors will also behave differently in different situations: in some circumstances strategic pricing mostly occurs after an OSS project has reached its steady-state size; whereas in others a commercial firm will aggressively confront an OSS project when it is small.

An open source movement is comprised of many individuals who contribute to the development, improvement, and debugging of the code, provide customer support, and/or decide to use what has been developed. Our main goal is to explore how outcomes at the community level are affected by features of the environment and by the motivations of the individuals involved. Without denying the relevance of traditional economic incentives, surveys of OSS communities typically report that other motivations are also important.⁴ Our approach is particularly influenced by the descriptions of Shah (2006). She 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 much smaller number of deeply involved participants.

Our formal model is of a software product which at any point in time 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 this naturally depreciates over time (perhaps because new needs such as compatibility with new hardware and software become more relevant). These declines can be offset by improvements made by a community of programmers. We assume that programmers 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 then decay over time. Without altruism, a programmer might develop a new feature, but would never publish it and make it available to other users. The community of

⁴See, for example, Lakhani and Wolf (2003) and Hertel et al. (2003).

altruistic programs may grow or shrink influenced by the quality of the software, and the quality of the OSS increases or decreases over time influenced by the number of altruistic programmers. In our baseline model, the programmers 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 some time until enough programmers join the community and become altruistic 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. The magnitudes of the strategic price distortions are more interesting. They depend on several factors including the loss in short-run profits from a price cut; the degree to which a price cut of given magnitude affects the state variables; and the persistence of changes in the state variables. We present some numerical examples, and note that distortions 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.

Section 5 extends the model to incorporate another feature of open-source communities that features prominently in descriptive papers: community members who lack the expertise to contribute code, but are actively involved in providing “customer support” to new users – they answer the many easy questions on bulletin board that expert programmers have neither the time nor the inclination to deal with. In particular, we add to our model a population of users and imbue them with similar altruism. Users can become altruistic for a time when the product meets their need, altruistic users provide customer support, and the fraction of users whose needs are met is assumed to be increasing in the amount of customer service that is being provided. Another new specification question that comes up is what motivates altruistic programmers: are they altruistic toward the code and value any contributions they make equally; or are they altruistic toward the user populations and value making contributions more when more users will take advantage of them. One result of this section is that the system dynamics are qualitatively different depending on the form of programmers’ altruism. Another is that the model can have “critical mass” effects. Whereas in our base model an OSS project can grow (albeit slowly) from any nonzero initial state, our model with customer service can be such that a substantial initial push of some type would be needed to prevent the project from collapsing to a zero-quality state.

The differences in dynamics will have implications for the behavior of competitors. When critical mass effects are present, a commercial competitor to an OSS project may want to set very low prices in the early stage of an OSS project’s lifecycle to drive it below critical mass, rather than mostly waiting until the project is more mature before pricing strategically.

As noted above, our paper is contributing to a literature on open source that is now substantial. Lerner and Tirole (2002) does a nice job of laying out the economics of contributions to open source. 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 Source-

Forge.net. Fershtman and Gandal (2007) also present data on contributions consistent with career-concerns motivations. Other papers provide evidence that other motivations are also relevant. Lakhani and von Hippel (2003) emphasize that helping others users provides direct learning benefits as well as costs. Hertel et al. (2003) and Shah (2004) note that a variety of benefits are mentioned in surveys including the simple enjoyment of participation that Stern (2004) argues is important for scientists.

Kuan (2001) considers a model of the OSS production function, where users contribute to the public good of the quality of the product. One feature it shares with the model in the latter part of our paper is that there are two types of users. Programmers contribute code and users report bugs. 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. Our model is related to these in that it 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).

Johnson and Myatt (2006) is another interesting paper that highlights mechanisms through which the open-source organizational form can be beneficial. It 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. The large number of individuals who can see and work with OSS code makes that type of collusion difficult to sustain in OSS projects.

Our analysis of competition between commercial and open source products follows at a smaller number of previous papers. Schmidt and Schnitzer (2003) discuss welfare issues in competition between open source and commercial software. 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. 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 and can be thought of as an endogenous network externality.

Our paper is also generally part of the recent literature on boundedly rational industrial organization.⁵ 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.

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 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:

⁵See 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).

| | |
|---------------------------|--|
| 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 <i>and</i> 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 Poisson 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.

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.⁶

3 Developer Behavior and Community Dynamics

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

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.

⁶Given 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.

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*

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

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 a 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 quality q_t and of the mass of programmers b_t are*

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

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

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

Then,

$$b_i(q) = \frac{\alpha\gamma_b q}{\alpha\gamma_b q + \delta}.$$

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

In words, $b_i(q)$ 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. This holds because the number of programmers who benefit from the software is high when q is high, and this will lead to an increase in the size of the altruistic population unless the altruistic group is already large enough to have sufficient depreciation. The function $b_i(q)$ is concave if this effect is less pronounced at higher levels

of quality. Note that the system has $\dot{b} > 0$ when $b < b_b(q)$ and $\dot{b} < 0$ when $b > b_b(q)$, so one can think of b as evolving toward the $\dot{b} = 0$ curve.

The system is in steady state where $b_q(\cdot)$ and $b_b(\cdot)$ 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_b and b_q curves in q - b space. The b_b curve is concave and the b_q is convex. If the b_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_b curve is steeper at the origin, then the fact that the b_b curve intersects the right side of the square (i.e. $b_b(1) \in (0,1)$) and the b_q curve intersects the top side of the square implies that there is an unique interior intersection. The right panel illustrates such a system.

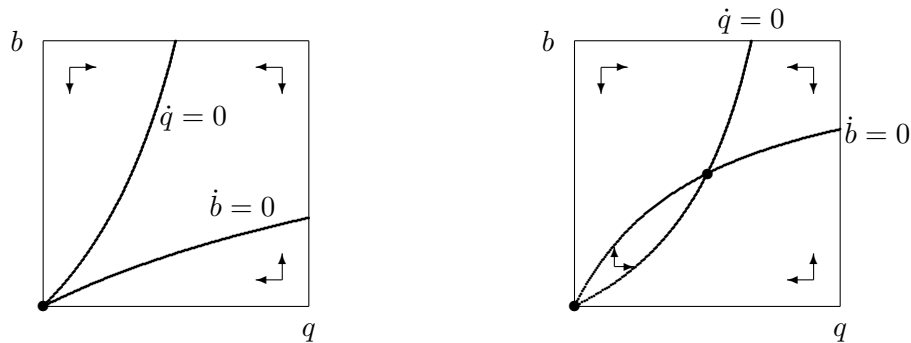


Figure 1: Model Dynamics: the left panel has $p = 0.5$ and $\gamma_b = \gamma_q = \beta = \delta = 1$. The right panel has $p = \gamma_b = \gamma_q = 1$ and $\beta = \delta = 0.5$.

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

$$b'_b(0) = \frac{\alpha\gamma_b}{\delta}$$

$$b'_q(0) = \frac{\beta}{\gamma_q}$$

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 frequency with which programmers encounter needs that can be met using the open-source product and consequent feelings of altruism must 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*

$$q^* = \frac{\alpha\gamma_b\gamma_q - \beta\delta}{\alpha\gamma_b\gamma_q + \alpha\beta\gamma_b}$$

$$b^* = \frac{\alpha\gamma_b\gamma_q - \beta\delta}{\alpha\gamma_b\gamma_q + \delta\gamma_q}$$

From here, we see that the steady-size of the project (and of the altruistic community) is increasing in $\alpha\gamma_b\gamma_q - \beta\delta$, the gap between opportunities 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 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 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 others have used it in the past.

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

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

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:

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

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_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

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

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 $[0, \bar{v}]$ for $\bar{v} > v + E$. In this case, the solution reduces to

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

yielding

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

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

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

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 monotonicity 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 strategic 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 factors: 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 set of parameters.⁷ 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 ($\frac{dq}{dp}$ and $\frac{db}{dp}$ 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- and dynamic-optimal pricing. The steady state size of the open source project, however, is somewhat lower with dynamic pricing.

⁷Values 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 $\lambda = 1$.

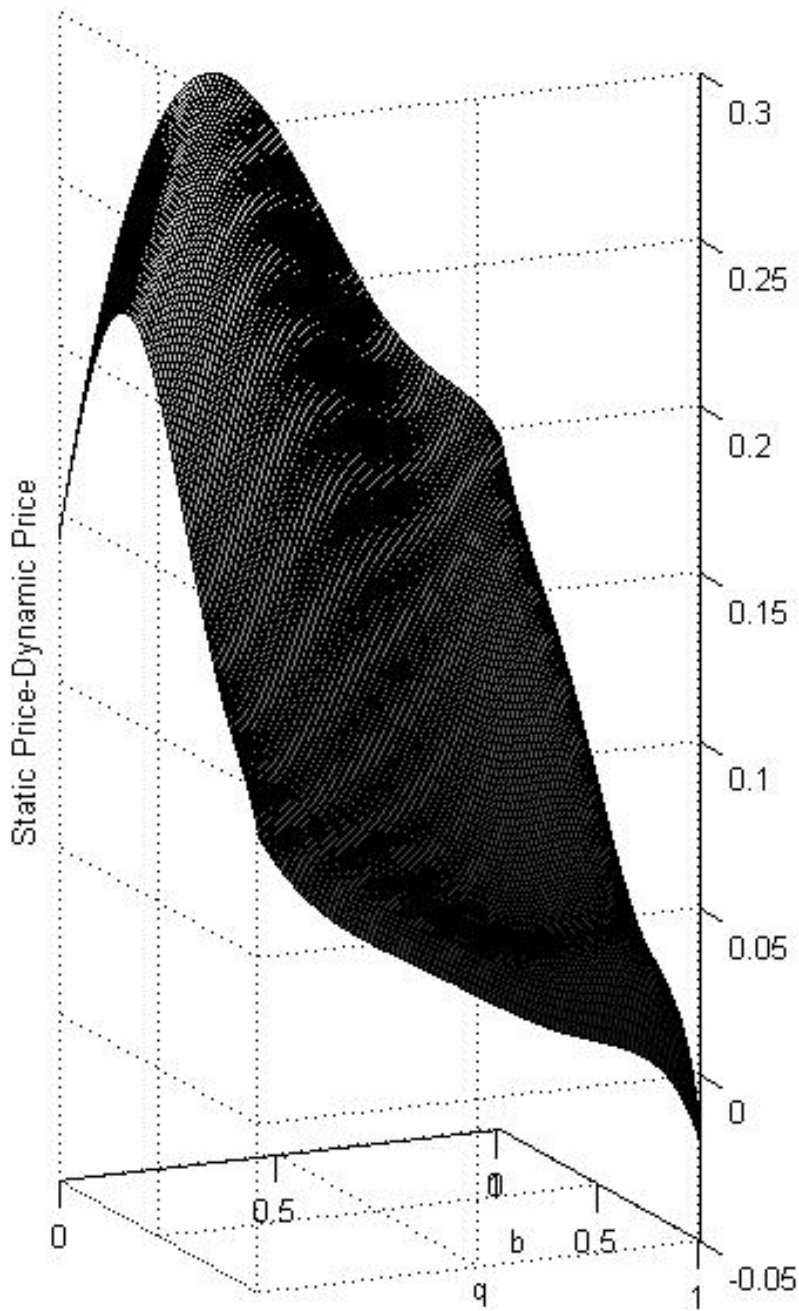


Figure 2: Strategic price distortions: the difference between the static and dynamic optimal price as a function of (q, b) for one set of parameters.

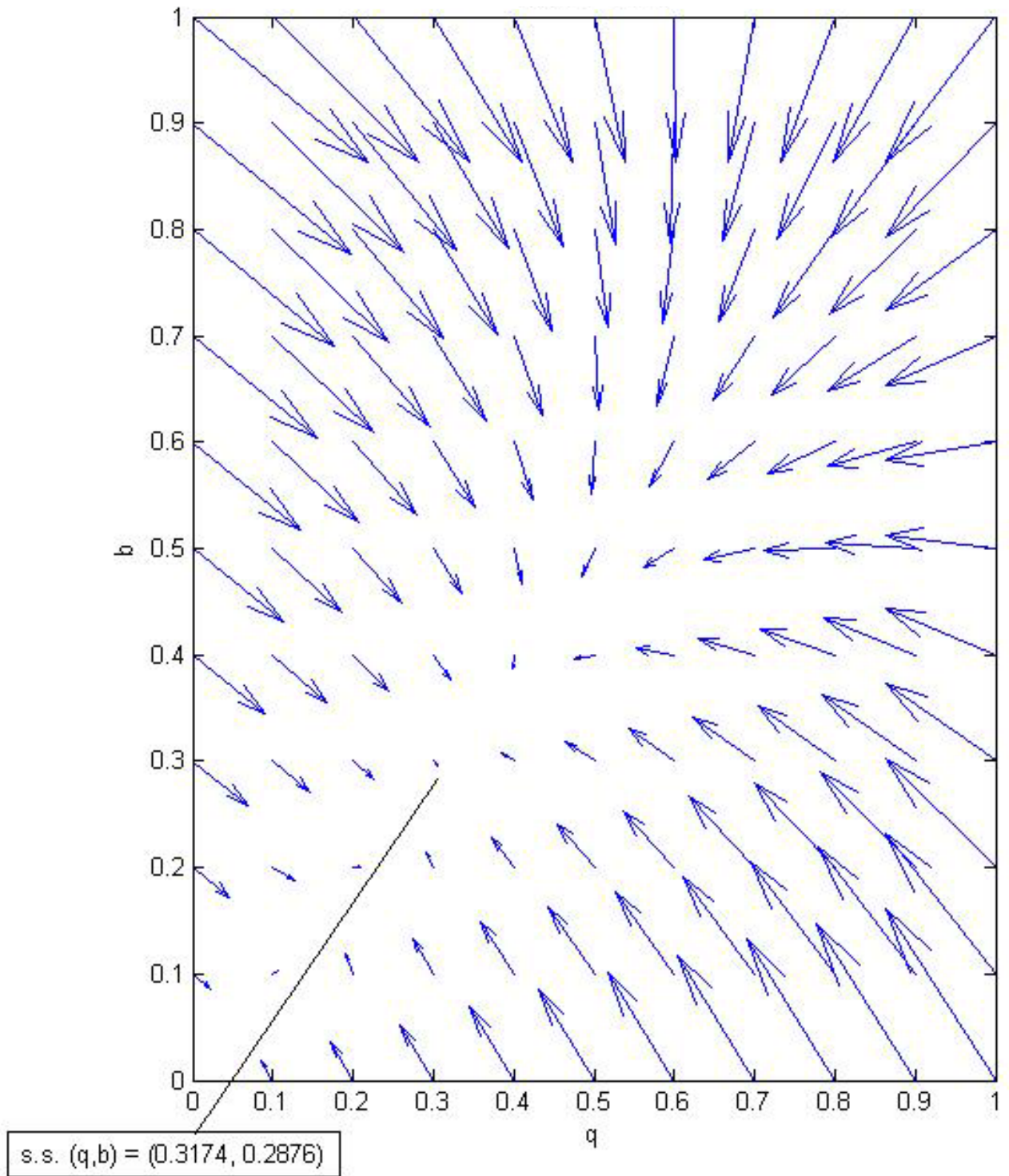


Figure 3: The vector field (\dot{q}, \dot{b}) of the system under static-optimal pricing.

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.⁸ 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, 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 $q_t f(m c_t)$, where c_t is the fraction of users who are altruistic at t and f is some concave function.⁹

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

⁸Again, Shah (2004, 2006) does a very nice job of providing descriptions and analysis.

⁹One 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.

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

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

From our analysis of the baseline model, we know that $(q_t, b_t) \rightarrow (q^*, b^*)$ 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

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

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 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

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.¹⁰ This suggests that a model should incorporate a relationship between altruism and the extent to which code is helpful to casual users. We develop such a model in this section 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 $m\gamma_u f(mc_t)$. If we assume that programmers 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 $am\gamma_u f(mc_t)$.¹¹

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 $[0, a]$.¹² This implies that the probability that an programmer decides to contribute a feature to the code base is $m\gamma_u f(mc_t)$.¹³

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*

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

¹⁰Shah (2006) quotes one programmer on this: “Why work on something that no one 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.

¹¹This 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.

¹²The 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 $\gamma_b = \gamma_q$.

¹³This assumes that the expression for the altruism benefit is always less than one.

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 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 $\gamma_{b'} = \gamma_b$, $\gamma_{q'} \equiv \gamma_q m \gamma_u f(m)$ and $\alpha' \equiv \alpha$. If we the

primitives of the model are such that $\alpha\gamma/b\gamma/q > \delta\beta$, 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.

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, e. g. by 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.

Although we have focused our discussion of strategic pricing on a commercial firm competing with open source, one could also bring out similar (albeit opposite) considerations for a firm that wishes to promote an OSS project. In particular, our model of user-motivated altruism provides one potential model to explain why some commercial firms have made very large short-term investments in supporting open-source projects.

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.

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 stylized; or one could leave the model as it is and take advantage of the tractability to add other considerations. 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. Another important consideration that we have avoided in this paper is

career concerns. While we felt that altruistic motivations were relatively understudied, we would not deny that career concerns are also relevant and could interact with altruistic motivations in interesting ways.

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 (2006): "Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows." *Management Science*, 52 (7), 1072-1084.
- [3] Della Vigna, Stefano and Ulrike Malmendier (2004): "Contract Design and Self-Control: Theory and Evidence," *Quarterly Journal of Economics*, 119, 353-402.
- [4] 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.
- [5] Eliaz, Kfir and Ran Spiegler (2008): "Consideration Sets and Competitive Marketing," mimeo, Brown University and London School of Economics.
- [6] Ellison, Glenn (2005): "A Model of Add-on Pricing," *Quarterly Journal of Economics*, 120, 585-637.
- [7] 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.
- [8] Ellison, Glenn and Sara Fisher Ellison (2009): "Search, Obfuscation, and Price Elasticities on the Internet," *Econometrica*, 77, 427-452.
- [9] Fershtman, Chaim and Neil Gandal (2007): "Open Source Software: Motivation and Restrictive Licensing," *International Economics and Economic Policy*, 4 (2), 209-225.
- [10] Gabaix, Xavier and David Laibson (2006): "Shrouded Attributes, Consumer Myopia, and Information Suppression in Competitive Markets," *Quarterly Journal of Economics*, 121, 505-540.
- [11] Heidhues, Paul and Botond Koszegi (2008): "Competition and Price Variation when Consumers are Loss Averse," *American Economic Review*, 98 (4), 1245-1268.
- [12] Johnson, Justin (2002): "Open Source Software: Private Provision of a Public Good," *Journal of Economics and Management Strategy*, 11 (4), 637-662.
- [13] Johnson, Justin and David P. Myatt (2006): "Collaboration, Peer Review and Open Source Software," *Information Economics and Policy*, 18 (4), 477-497.
- [14] 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.

- [15] Kamenica, Emir (2008): “Contextual Inference in Markets: On the Informational Content of Prouct Lines,” *American Economic Review*, 98 (5), 2127-2149.
- [16] Lakhani, Karim and Eric von Hippel (2003): “How Open Source Software Works: “Free” User-to-User Assistance,” *Research Policy*, 32 (6), 923-943.
- [17] 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.
- [18] Lerner, Joshua, Parag Pathak and Jean Tirole (2006): “The Dynamics of Open Source Contributors,” *American Economic Review Paper and Proceedings*, 96 (2), 114-118.
- [19] Lerner, Joshua and Tirole, Jean (2002): “Some Simple Economics of Open Source,” *Journal of Industrial Economics*, 50 (2), 197-234.
- [20] Lerner, Joshua and Tirole, Jean (2005a): “The Scope of Open Source Licensing,” *Journal of Law, Economics, and Organizations*, 21, 20-56.
- [21] Lerner, Joshua and Tirole, Jean (2005b): “The Economics of Technology Sharing: Open Source and Beyond,” *Journal of Economic Perspectives* 19 (2), 99-120.
- [22] Rabin, Matthew (1993): “Incorporating Fairness into Game Theory and Economics,” *American Economic Review*, 83 (5), 1281-1302.
- [23] Raymond, Eric S. (2001): *The Cathedral and the Bazaar*, O’Reilly.
- [24] 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.
- [25] 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.
- [26] Shah, Sonali (2006): “Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development.” *Management Science*.
- [27] Spiegler, Ran (2006a): “The Market for Quacks,” *Review of Economic Studies*, 73 (4), 1113-1131.
- [28] Spiegler, Ran (2006b): “Competition over Agents with Boundedly Rational Expectations,” *Theoretical Economics*, 1 (2), 207-231.
- [29] Stern, Scott (2004): “Do Scientists Pay To Be Scientists?,” *Management Science*, 50 (6), 835-853.
- [30] Tirole, Jean (1988): *The Theory of Industrial Organization*. Cambridge MA: MIT Press.