View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by AIS Electronic Library (AISeL)

CODE ARCHITECTURE AND OPEN SOURCE SOFTWARE DEVELOPMENT

Completed Research Paper

Gang Peng
Williamson College of Business
Youngstown State University
One University Plaza
Youngstown, OH 44555
gpeng@ysu.edu

XianJun Geng
School of Management
University of Texas at Dallas
800 W Campbell Road, SM 33
Richardson, TX 75080
geng@utdallas.edu

Lihui Lin
School of Management
Boston University
595 Commonwealth Avenue
Boston, MA 02215
lhlin@bu.edu

Abstract

A model is developed to study how the code architecture affects open source software (OSS) development. The model incorporates the resource heterogeneity and diverse motivations of various groups of programmers as well as the strategic interactions among them. We argue that the major advantage brought by a modular architecture of OSS code base is that it reduces both the cognitive cost and the coordination cost associated with OSS development, thus allowing programmers more easily to locate, manage, and contribute to the code base. We show that in OSS development, while modular architecture can potentially increase code contribution, it does not necessarily reduce free-riding; in fact it may well increase free-riding due to the strategic interactions among the programmers. We further empirically test the predictions using the SourceForge OSS development data, and the results confirm our theoretical predictions. The findings bear important theoretical as well as practical implications and provide guidelines for practitioners of OSS development and the collective innovation in general.

Keywords: Open source software development, modular architecture, code contribution, free-riding

Introduction

Open source software (OSS) is generally considered to be public goods since the software can be distributed and further incorporated freely into any other software through OSS licenses (e.g., von Hippel and von Krogh 2003; Baldwin and Clark 2006). The huge success of OSS projects like Linux, Apache, MySQL, Perl, PHP, and Mozilla has made many believe that the OSS development model could potentially revolutionize the innovation process in much broader research areas (e.g., von Hippel and von Krogh 2003; Rai 2005; Shah 2006; Fleming and Waguespack 2007).

However, successful application of OSS model still faces many challenges (von Krogh and von Hippel 2006; von Krogh and Spaeth 2007). One of the puzzling questions is the motivation for programmers to contribute to the code base. Under traditional software development model, programmers are hired and collocated to code the software projects. But in OSS model, programmers are not paid and moreover, the software is available to anybody for free use (Raymond 1999), thus why programmers are willing to participate in the development of OSS has puzzled many researchers. However, recent studies have revealed that programmers do not really contribute for free; rather they contribute out of a set of complex and intertwined motivations, and they recoup certain private benefits as well as contributing to the success of OSS projects as a whole (e.g., Roberts et al. 2006; Bagozzi and Dholakia 2006). Consequently, researchers have proposed that OSS model is a private-collective innovation model and represents a very rich and fertile middle ground where incentives for private investment and collective action can coexist (von Hippel and von Krogh 2003; Haefliger et al. 2008).

A related problem is free-riding or inequality of code contribution to OSS development: although many projects can potentially attract large number of programmers, quite often only a small number of individuals code the software (Fitzgerald 2004; Kuk 2006). If not properly addressed, free-riding can derail the development of OSS projects (e.g., Raymond 1999; Baldwin and Clark 2006), and will endanger the application of OSS model to other areas as well. Thus how to promote participation and reduce free-riding is of strategic importance not only to OSS development but also to the success of open innovation in general (Chesbrough 2003; 2007).

Recent studies proposed that modular code architecture can potentially reduce free-riding in OSS development (e.g., Johnson 2002; Baldwin and Clark 2006). In a modular design, the whole software code base is divided into several loosely coupled modules, and consequently the code structure is transformed from a monolithic architecture into a modular architecture (Parnas 1972). The responsibility for developing each module is delegated to specific individuals (Mockus et al. 2002), so participation tends to be equalized under the modular architecture (Johnson 2002; Baldwin and Clark 2006). However, prior studies on modular OSS architecture have several limitations. First, OSS is typically considered as a pure public good (e.g., Johnson 2002; Baldwin and Clark 2006), and virtually none have incorporated the private aspect of OSS development. Second, code contribution by participants is studied as a binary variable, i.e., programmers either contribute or not contribute to the software, and none have considered the differences in the quantity of contribution. However, in reality, contribution can be better quantified as a continuous variable not a binary variable. Third, prior studies do not incorporate participants' diverse motivations and the resource endowments, which play a critical role in their reaction to the design of code structure. Fourth, there is virtually no empirical evidence to support the effect of modular architecture on OSS development. In this study we intend to fill these research gaps.

Research Context

Innovations are critical for the competitiveness of firms and nations (Cooper 1993; Dougherty and Hardy 1996; Penrose 1995). Traditionally, two models of innovation have been practiced: private model and collective model. In the private model, innovators invest private resources to develop the innovation, and in return, they own the innovation produced and collect private returns typically through property rights (Dam 1995). In the collective innovation, the innovators usually are subsidized to develop the innovation and they relinquish control of innovation developed and make it a public good by unconditionally supplying the innovation to a "common pool" (Liebeskind 1996; Osterloh and Rota 2007). However, OSS model deviates from both these two models in that innovators invest their own private resource for the development of a public good, thus OSS model potentially represents a new type of innovation (von Hippel and von Krogh 2003). As commented by prior studies that contributions to open source software development are not pure public goods—they have significant private considerations even though the innovation will be freely revealed, thus OSS has been termed as a private-collective model of innovation (von Hippel and von Krogh 2003).

Over time, researchers have proposed a series of inter-related intrinsic and extrinsic motivations for OSS contribution from different perspectives such as self use or "personal itch" (Raymond 1999; von Hippel and von Krogh 2003), affiliation and identity (Hertel et al. 2003), signaling and career concern (Lerner and Tirole 2002), peer recognition (Hars and Ou 2002; Lakhani and von Hippel 2003), reputation and status (Roberts et al. 2006), and hedonic motives such as enjoying programming (Hertel et al. 2003; Shah 2006), etc. Three observations from these facts can be derived. First, rather than contributing for free, programmers do derive benefits from OSS code contribution. Even though programmers do not gain immediate benefits, they potentially gain delayed benefits in the future (Lerner and Tirole 2002). Second, although OSS developers derive a variety of benefits, they are either private benefits such as fun, knowledge, and career concern, etc., or public benefit, i.e., the delivery of the final OSS projects. Third, whether participants derive private benefits or public benefits depends critically on the roles they play during the OSS development process.

Literature suggests that participants in OSS community mainly fall into two types, the hobbyists who derive benefits from its own coding activities, and the need-driven participants who use the OSS for their own purposes (e.g., Shah 2006). Hobbyists are also referred as hackers in literature (von Hippel and von Krogh 2003), and we will use this name hereafter. By contributing to OSS code, hackers derive fun and knowledge from their contribution, and gain peer recognition and reputation (Raymond 1999; Stewart 2005), and even signal their talents to attract prospective employers and venture capital (Lerner and Tirole 2002).

A closer examination of the roles played by the need-driven participants reveals that they can be further divided into two subtypes: project leaders and onlookers. Project leaders are typically those who initiate the projects trying to solve their own problems (von Hippel and von Krogh 2003), and they set up virtual workspace and provide initial code bases for project development (Shah 2006). The primary motivation for leaders is the delivery of the final software (Raymond 1999; von Hippel and von Krogh 2003). Onlookers are the programmers who join the projects after the project is initiated. Similar to the leaders, their primary concern is also the final software, but different from the hackers they do not want to contribute unless they have to, e.g., when the OSS project is in slow progress or to be abandoned due to lack of contribution. This type of programmers corresponds to the "triangle contributors" in the provision of public goods (Fischbacher and Gachter 2010).

One of the noticeable trends for OSS development is that the code architecture becomes increasingly modular. Although modular architecture is well-known in software development, the concept of modular design extends beyond software industry. In fact, modular code architecture falls under the larger picture of modular product design. In general, in a complex system which is made up of large number of components that interact in a non-simple way, a modular design can group the components into a smaller number of subsystems so to reduce the interdependency between each component (Langlois 2002). Software development is one of the areas that have witnessed the most mature application of modular product design (Fixson 2007). A software module captures a set of design decisions which are hidden from other modules, and modules interact with each other primarily through their interface, thus modular design promote encapsulation or information hiding by separating a module's interface from its implementation (Parnas 1972). The benefits of modular architecture include concurrent development, robust to interruption of the production process, reduced communication cost, recombination of source code, code reuse, and increased quality, etc. (Gershenson et al. 2003; Haefliger et al. 2008). In addition to the above-mentioned benefits, researchers have also proposed that modular code architecture is an effective mechanism to alleviate free-riding in OSS development, and in this regards, two most important studies are Johnson (2002) and Baldwin and Clark (2006), who argue that modular code architecture can significantly reduce free riding.

While path-breaking and insightful, the work of Johnson (2002) and Baldwin and Clark (2006) leave room for improvement. We feel that some of the important and unique aspects of OSS development need to be incorporated when studying the impact of code architecture: First, as discussed earlier, contributors are heterogeneous in terms of their motivations and resource endowment, which will play critical roles when contributors react to the design of the code architecture (Raymond 1999). Second, code contribution is a collective effort, thus OSS code contribution model should be flexible enough to accommodate contribution made concurrently and differently in quantity, not as assumed that a single module is coded only by one programmer and only once (Johnson 2002; Baldwin and Clark 2006). Third, we believe that it is important to distinguish between two concepts in studying OSS development: code participation and code contribution. Code participation is measured as a binary variable, either to code the project or not (Baldwin and Clark 2006). In contrast, code contribution is a continuous variable, and it measures how much one codes the projects in terms of lines of code (LOC) or commits made to the code base. In other words, code contribution is a more accurate measurement of free-riding than merely participating in the coding activities. Incorporating the above aspects will shed new light on how code architecture affects code contribution in OSS development.

The Model

Programmers working on an OSS project incur a variety of costs. By engaging in an OSS project, programmers lose their time to work on other programming tasks, such as their own proprietary projects (Lerner and Tirole 2002; Shah 2006).[1]

Suppose there is one OSS project which enrolls *N* programmers, including leaders, onlookers, and hackers. Beside the OSS project, programmers also engage in their own proprietary projects. Programmer *i*, *i=1,2,...,N*, is endowed with a total available work time of w_i. Programmer *i* will produce code $p_i(x_i)$ if she spends $x_i \geq 0$ units of time on her proprietary projects, and produce code $o_i(g_i)$ if she spends $g_i \geq 0$ units of time to develop the OSS project. The total supply of time to the OSS project by all programmers is denoted by $G = \sum_N g_i$. Further assume that the codes provided by each programmer to the OSS project are additive, i.e., the total amount of code provided by all the programmers is given by $O = \sum_N o_i(g_i)$, which will be shared by all programmers despite their actual levels of contribution. As discussed earlier, depending on their types, programmers possibly derive utilities from three sources: code contribution to

[1] A proprietary project a programmer works on can be either for work use or for personal use.

their own projects $p_i(x_i)$, their own parts of OSS contribution $o_i(g_i)$, and the whole OSS code O, thus the utility function of programmer i is given by $v_i[p_i(x_i), o_i(g_i), O]$, where v_i is the utility function increasing in all three parameters. To simplify the analysis, assume the code production function for all programmers are the same, i.e., $o_i(g_i) = o(g_i)$, and $o(g_i)$ is linear and increasing in g_i, then $O = \underset{N}{O}[o(g_i)] = \sum_N o(g_i) = o(\sum_N g_i) = o(G)$. Since programmers maximize their utilities, programmer i will solve the following problem:

$$\begin{aligned} \max_{x_i, g_i} \quad & u_i(x_i, g_i, G) \\ s.t. \quad & x_i + g_i \le w_i \\ & G_{-i} + g_i = G \\ & g_i \ge 0 \end{aligned} \tag{1}$$

where u_i is the transformed utility function and G_{-i} is the total contribution by all other programmers except programmer i. It can be verified that u_i is non-decreasing in all three parameters as well. Equation (1) shows that there is a tradeoff between time allocated for programmer *i*'s propriety project and the OSS project: the same amount of time devoted for propriety project will be sacrificed for developing the OSS project.

Equation (1) has several advantages over the model presented in prior studies. First, it extends the scope of prior studies from merely code participation to code contribution. Second, it allows multiple programmers to code the same module thus more closely reflect real programming practices. Third, it incorporates the facts that individual programmers are constrained by their resource endowment (w_i), and, when considering code contribution, they need to allocate resources between two alternative activities. As in Johnson (2002) and Baldwin and Clark (2006), the model retains the strategic interactions among programmers when deciding on their own contribution level.

In OSS development, modular code architecture can induce or facilitate code contribution through various mechanisms: First, a modular structure reduces the cognitive cost of the potential contributors. Modular design splits the whole project into loosely coupled components, which have less complexity (von Hippel 1990), and can be more easily identified and managed. Thus smaller and less complex code components decrease the quantity of information needs to be processed. Indeed, modular design is consistent with the "divide-and-conquer" philosophy in software engineering (e.g., Bentley 1980; Lenat 1995). Second, a modular structure reduces the coordination cost among the project participants. Due to the less tightly-coupled nature, a modular design allows one to contribute with less concern of the codes in other modules, thus different modules can be developed concurrently and then integrated later (MacCormack et al. 2006). Consequently, a modular design significantly cut down the coordination cost. Third, because modules within the code base do not talk to each other directly, code modifications made within one module do not interfere the functioning of other modules, and code produced using modular architecture tends to have less bugs and higher quality. Fourth, a modular architecture makes it easier for potential contributors to identify the code components that are most interesting to them, thus can facilitate the code development. This is particularly important for OSS development, since programmers usually assign themselves to the tasks rather than being assigned by the project leaders as in proprietary software development teams (Shah 2006).

However, modular code architecture has its negative side as well. As discussed earlier, some of the most cited reasons for OSS contributions are reputation and status, peer recognition, careen concern, etc., all of which will be more easily achieved when a larger group of peers are involved. However, a modular architecture may run counter to this goal. Prior literature on modular design has shown that there exists a mirroring or mapping between the product architecture and the team or organizational structure that is adopted to manufacture the product (Henderson and Clark 1990; Schilling 2000). Thus a modular code architecture tends to lead to a modular team structure that consists of multiple self-contained and independent groups of programmers who will end up interact with few peers in smaller groups. Therefore,

as code architecture gets increasingly modular, ceteris paribus, programmers will have less incentive to contribute to the code base due to reduced number of observable peers.

To incorporate the effect of modular code architecture, we use variable $m \geq 0$ to measure the level of modularity of the code structure: a higher m corresponds to a more modular design and vice versa. Thus, Equation (1) can be modified as the following:

$$\begin{aligned} &\max_{x_i, g_i} \; u_i(x_i, g_i, G, m) \\ &s.t. \quad x_i + \frac{g_i}{m} \leq w_i \\ &\qquad G_{-i} + g_i = G \\ &\qquad g_i \geq 0 \end{aligned} \tag{2}$$

Since $g_i \leq (w_i - x_i)m$, variable m in fact determines the *effective* time that can be spent on the OSS project for any units of time taken away from developing the proprietary project. If m = 1, then Equation (2) is the same as Equation (1). However, if m>1, then for any units of time taken away from developing the proprietary project, more *effective* units of time can be devoted for the OSS project. This is what we would expect from a modular design: as the OSS code architecture gets more modular, code contribution will be more efficient and less costly. The situation will be reversed if m<1. Thus, parameter m captures the code architecture for this code contribution game. The negative side of a modular architecture is captured in the utility function. As before, $\partial u_i / \partial g_i \geq 0$, however, we further specify that $\partial^2 u_i / (\partial g_i \partial m) < 0$ to reflect that as code architecture gets increasingly modular, the marginal utility of code contribution decreases.

Next we incorporate the three types of OSS programmers discussed earlier and study how a modular design could potentially affect their respective code contributions. As discussed earlier, leaders, onlookers, and hackers attach different importance to individual contribution g_i and the whole OSS source code G. First, prior literature has shown that project leaders attach more importance to the OSS project than others (Lerner and Tirole 2002). In contrast, hackers derive utilities mainly from their own fun, knowledge, and reputation gained from coding their parts of OSS source code, as well as their own proprietary projects. Thus, we use parameter α_i to differentiate these three types of programmers, and correspondingly, we modify Equation (2) to take the following form:

$$\begin{aligned} &\max_{x_i, g_i} \; u_i = \alpha_i S(g_i, m) + (1 - \alpha_i) T(G) + x_i \\ &s.t. \quad x_i + \frac{g_i}{m} \leq w_i \\ &\qquad G_{-i} + g_i = G \\ &\qquad g_i \geq 0 \end{aligned} \tag{3}$$

where $S(\cdot)$ is the utility function from private benefits and $T(\cdot)$ is the utility function from public benefits. We specify $0 < \alpha_i \leq 1$, so if α_i is small (yet not zero), it's a leader; if α_i is large (yet not one), it's an onlooker; if $\alpha_i = 1$, it's a hacker. As before, $S_{g_i} > 0$, $T_G > 0$, while $S_{g_i g_i} < 0$ and $T_{GG} < 0$. However, $S_{g_i m} < 0$, $S_{g_i mm} < 0$, and $S_{g_i g_i m} < 0$.

Equation (3) can be further simplified as:

$$\max_{g_i} \; u_i = \alpha_i S(g_i, m) + (1 - \alpha_i) T(\, g_i + G_{-i}) + (w_i - \frac{g_i}{m}) \tag{4}$$

Then we can obtain the following propositions from Equation (4).

***Proposition 1*:** For the code contribution game in Equation (4), there exists a unique Nash Equilibrium.

Proof: From Equation (4), we have $u_i = \alpha_i S(g_i, m) + (1-\alpha_i)T(g_i + G_{-i}) + (w_i - g_i / m)$. The FOC is $\partial u_i / \partial g_i = \alpha_i S_{g_i} + (1-\alpha_i)T_G - 1/m = 0$. Suppose we have inner solutions, and the total contribution at equilibrium is G^*, then at equilibrium, programmer *i*'s optimal contribution g_i^* is determined implicitly by:

$$S_{g_i^*} = \frac{1/m - (1-\alpha_i)T_{G^*}}{\alpha_i} = T_{G^*} + \frac{1/m - T_{G^*}}{\alpha_i} \tag{5}$$

Suppose Equation (5) can be solved explicitly, so $g_i^* = r_i(G, m, \alpha_i)$, then it follows from Equation (5) that r_i is decreasing in G^*. Let $G = R(G, m, \alpha) = \sum_i r_i$, then R is decreasing in G^* as well. Graphically, the equilibrium will be reached when R intercepts with the 45° line as shown in Figure 1, and this is unique. Thus there is a unique equilibrium for this code contribution game. □

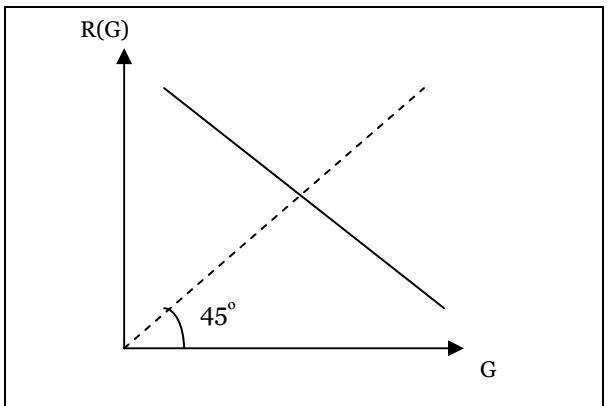

Figure 1. Existence of Unique Nash Equilibrium

Proposition 2: Let g_i^* be the code contribution of a current contributor *i* at equilibrium, then g_i^* is increasing in the order of leaders, onlookers, and hackers.

Proof: Again, this follows from Equation (5). Since S_{g_i} and T_G are decreasing functions ($S_{g_i g_i} < 0$ and $T_{GG} < 0$), therefore, for a given total contribution at equilibrium G^*, g_i^* increases in α_i. In other words, in terms of contributions to the OSS development, leaders < onlookers < hackers. □

This finding is very interesting: the ones who hold the most interests in the project contribute nevertheless the least. The leaders, anticipating the contributions from the hackers and the onlookers who derive more private utility from their contribution, strategically hold back their efforts in this code contribution game.

Proposition 3: Let *m* be the number of modules an OSS project has, then there exists a threshold *m**, such that $\frac{dg_i^*}{dm} > 0$ if *m*<*m**, i.e., modular architecture will increase the individual

contributions from all programmers if $m<m^*$. In parallel, if G^* is the total code contribution from all programmers, then it follows $\frac{dG^*}{dm} > 0$ if $m<m^*$, i.e., modular architecture will increase the total contributions from all programmers to the extent that $m<m^*$.

Proof: From Equation (5), we have $y(g_i, m) = 1/(\alpha_i m) - (1/\alpha_i - 1)T_{G^*} - S_{g_j^*} = 0$, so:

$$\partial g_i^* / \partial m = -\frac{\partial y / \partial m}{\partial y / \partial g_i^*} = -\frac{-1/(\alpha_i m^2) - S_{g_i^* m}}{-S_{g_i^* g_i^*} - (1/\alpha_i - 1)T_{G^* G^*}} = \frac{1/m^2 + \alpha_i S_{g_i^* m}}{-\alpha_i S_{g_i^* g_i^*} - (1-\alpha_i)T_{G^* G^*}} \tag{6}$$

Since the denominator $-\alpha_i S_{g_i^* g_i^*} - (1-\alpha_i)T_{G^* G^*} > 0$, then $\partial g_i^* / \partial m > 0$ only if $1/m^2 + \alpha_i S_{g_i^* m} > 0$, from which we obtain $m^* = \min\{m_i^* = 1/(-\alpha_i S_{g_i^* m})^{1/2}\}$, and it follows that $\partial g_i^* / \partial m > 0$ for all programmers if $m<m^*$. As $G = \sum_N g_i$, it follows that $\frac{dG^*}{dm} > 0$ as well if $m<m^*$. □

Corollary 1: Let *Pert* be the percentage of positive contributors in an OSS project, then $\frac{dPert_i^*}{dm} > 0$ if $m<m^*$, i.e., as the code structure becomes more modular, the percentage of programmers who contribute positively to the code base will increase if $m<m^*$. But as m increases above m^*, the percentage will decrease: first the hackers will stop contributing, followed by the onlookers, and finally the leaders.

Corollary 1 follows directly from Proposition 3, since m_i^* increases in the order of hackers, onlookers, and leaders. Proposition 3 and Corollary 1 reveal that modular code architecture can potentially reduce inequality in coding activities in terms of both the actual codes contributed and the number of programmers who participate in the coding activity. This is consistent with the finding of Baldwin and Clark (2006).

Proposition 2 and 3 reveals the effects of modular architecture on the actual levels of code contribution for the OSS projects, but not on free-riding *per se*, which is a measure of relative inequality of contribution. However, as discussed earlier, free-riding is better measured by contribution rather than participation. To be specific, we refer free-riding in our study as inequality of code contribution, not merely code participation as in prior studies (e.g., Johnson 2002; Baldwin and Clark 2006). There are various measures for inequality of resources or contributions in literature, and probably the best known is *Gini* coefficient, which is widely used to measure income inequality and is defined as one-half of the relative mean difference, the arithmetic average of the absolute values of differences between all pairs of values (Sen 1973). Consistent with prior studies on OSS code contribution (e.g., Kuk 2006), we use *Gini* coefficient to measure the extent of programmers' free-riding within a project:

$$Gini = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\left|g_i - g_j\right|}{2N^2\bar{g}} \tag{7}$$

where g_i (or g_j) is the contribution made by programmer *i* (or *j*), $\overline{g}$ the average contribution made by all programmers, and N is the number of programmers on the project (no matter they contribute or not).[2] It can be shown that $Gini \in [0,1]$, and the higher the value of *Gini*, the more severe the free-riding in a group (Sen 1973). If *Gini* = 0, then every programmer contributes the same, and if *Gini* = 1, only one programmer contributes and the rest do not. Next, we introduce a proposition about how modular code architecture affects free-riding:

Proposition 4: Let *Gini* coefficient be the measurement of free-riding, then $\frac{dGini}{dm} \geq 0$ if $m < m^*$ and $\frac{dGini}{dm} < 0$ if $m \geq m^*$, i.e., modular code architecture has a curvilinear relationship with free-riding.

Proof: If we order g_i in ascending order, then it can be shown that Equation (7) is equivalent to the following:

$$Gini = \frac{2}{N^2 \overline{g}} \sum_{i=1}^{N} i(g_i - \overline{g}) \tag{8}$$

From Proposition 2, at equilibrium, programmers contribute in the order of leaders, onlookers, and hackers. Thus a sufficient condition for Proposition 4 to be true is that, for $m < m^*$, g_i^* increases fastest for hackers, followed by the onlookers, and then the leaders as code architecture becomes increasingly modular; and this trends, however, is reversed for $m \geq m^*$. This is equivalent to say that for $m < m^*$,

$$\frac{\partial^3 g_i^*}{\partial m^2 \partial \alpha_i} > 0 \tag{9}$$

and it is the opposite for $m \geq m^*$. To proceed, from equation (6), we obtain:

$$\frac{\partial^2 g_i^*}{\partial m^2} = \frac{(2/m^3 - \alpha_i S_{g_i^* mm})[\alpha_i S_{g_i^* g_i^*} + (1-\alpha_i) T_{G^* G^*}] + (1/m^2 + \alpha_i S_{g_i^* m})\alpha_i S_{g_i^* g_i^* m}}{[\alpha_i S_{g_i^* g_i^*} + (1-\alpha_i) T_{G^* G^*}]^2} \tag{10}$$

It follows from Proposition 3 that $1/m^2 + \alpha_i S_{g_i^* m} > 0$ if $m < m^*$. Therefore $\frac{\partial g_i^*}{\partial m} > 0$ and $\frac{\partial^2 g_i^*}{\partial m^2} < 0$ if $m < m^*$, and g_i^* is a concave and increasing function of m. It can be further shown that given $S_{g_i g_i} < T_{GG}$, $\frac{\partial^3 g_i^*}{\partial m^2 \partial \alpha_i} > 0$. Thus, when $m < m^*$, g_i^* increases fastest for the hackers, then the onlookers, and last the leaders. Therefore, it follows $\frac{dGini}{dm} \geq 0$ if $m < m^*$. However, from Corollary 1, once m passes the critical level of m^*, hackers will start to decrease their contribution while onlookers and leaders still increase their contribution; this will narrow down the gaps between code contribution, and therefore it follows from Equation (8) that *Gini* will start to decrease for $m \geq m^*$. □

[2] In this study, we use two measures of code contribution to calculate *Gini* coefficient: the commits made and the lines of code (LOC) contributed by programmers.

Therefore, Proposition 4 shows that *Gini* is a concave function of m overall, and contrary to what was believed that a modular architecture will always reduce free-riding, modular architecture could potentially increase free-riding due to the strategic interactions among the programmers.

Empirical Evidence

To test the effect of code architecture on OSS development, we make use of a dataset obtained from SourceForge.net, the world's largest OSS project-hosting website. At SourceForge, project leaders recruit other members and grant them access to code base so they can contribute source code to the projects. SourceForge provides rich information about the software project such as project characteristics, developer characteristics, developer roles, and activities at project forums and bug tracking service, etc. To study how code structure affects code contribution, we make use of the projects' *CVS* log file. At SourceForge, programmers make changes to the source code through the Concurrent Versioning Systems or *CVS* (Fogel 2006), and each batch of changes made to the source code is called a commit, and is recorded in the *CVS* log file. Within each commits, the *CVS* also records the lines of code (LOC) that are changed. The *CVS* log file provides an excellent source of data to study the software development activities, since it enables us to trace exactly who contribute what to which module of the project and when.

At SourceForge, projects are categorized into foundries, or groups of projects that share a common programming language. As of May 2006, there are over 80,000 projects registered at SourceForge, thus we restrict our sample only to the Java foundry (Grewal et al. 2006). We further restrict our samples to the projects that have at least two programmers so to study the free-riding problem. To avoid left censoring problem, we only examine projects registered on and after January 1, 2003, so we can observe the complete coding history for each of these projects until May 2006. This leaves us 517 projects in the final sample.

For each of the project, we identify the number of modules a project has, the age of the project (in months) till May 2006, the programmers in each project, and coding activities of each programmer, as well as other project characteristics such as intended audience, operating systems, and topics. In addition, we record the number of commits and LOC made by each programmer within a project. The variable definitions are provided in Table 1.

We first examine the descriptive statistics and correlation matrix (omitted due to page limitation). The two measures of *Gini* coefficient are highly correlated, indicating good internal validity of both measures. The same is true for the two measures of code contribution.

Table 1. Definition of Variables

Variables	Definition
*Gini*_commit	Gini coefficient for measuring free-riding for code contribution as calculated by Equation (7). It is based on commits made by each programmer.
*Gini*_LOC	Gini coefficient for measuring free-riding for code contribution as calculated by Equation (7). It is based on LOCs made by each programmer.
Code_commit	The log of the total number of commits programmers contributes to the source code within an OSS project.
Code_LOC	The log of the LOCs programmers contributes to the source code within an OSS project.
Pert	The percentages of the positive contributors among all programmers enrolled for an OSS project.
Module	The number of modules a project has, divided by 100.
Project size	The total number of developers on the project.

To examine the coding activities of programmers, we need to identity the leaders, onlookers, and the hackers. SourceForge dataset only identifies the leaders, not onlooker and hackers. However, SourceForge had a survey for all programmers in January 2003. Three questions are helpful to identify the likelihood that participants are willing to help others. They are: 1) "How strongly do you believe in Open Source Software?", 2) "How willing are you to answer support questions about open source project?" and 3) "If your skills match, how interested are you in helping developers from other projects?". Each of the questions is scored from 1 to 5. We consider those having an average score of 4 or higher as hackers if they are not leaders. Then we list the contribution from the three types of programmers in Table 2. It shows that leaders contribute less than the onlookers who further contribute less than the hackers, thus we conclude that Proposition 2 is supported.

Table 2. Programmer Constituents and Average Code Contribution

	Average Number	percentages	Average commits	Average LOC
Leaders	1.63	44%	98.65	5,732
Onlookers	1.28	35%	265.18	32,222
Hackers	0.76	21%	515.85	16,515

Proposition 3 predicts that as the code architecture gets more modular, total contribution will first increase and then decrease. We use the following OLS model to test Proposition 3:

$$Contribution = \alpha_1 Size + \alpha_2 Module + \alpha_3 Module^2 + \boldsymbol{\beta} Characteristics + \varepsilon \quad (11)$$

where $Contribution$ is the total code contribution of a project team, α s are scalars, $\boldsymbol{\beta}$ is a vector, and ε is the error term. The heteroskedasticity robust results are shown in Table 3. Model 1 and 3 do not include the research variables *Module* and *Module*^{2}. However, the explanatory power increases significantly in Model 2 and Model 4 when *Module* and *Module*^{2} are included. Moreover, in both Mode 3 and 4, the coefficient on *Module* is positive and significant, while that on *Module*^{2} is negative and significant. These results show strong support for Proposition 3.

Table 3. Estimation Results for Proposition 3

Independent Variables	Model 1	Model 2	Model 3	Model 4
Project size	0.262*** (0.026)	0.100*** (0.019)	0.269*** (0.027)	0.106*** (0.023)
Module		4.913*** (0.499)		4.920*** (0.532)
Module^2		−1.392*** (0.332)		−1.384*** (0.324)
R^2	0.350	0.641	0.308	0.523

Notes: N=517. ***p<0.01. Dependent variable is the Code_commit in Model 1 and 2, and Code_LOC in Model 3 and 4. Estimated coefficients and their associated standard errors (in parentheses) are listed under each equation. Other control variables include 34 dummies for project characteristics such as development stages, intended audience, operating systems, and project topics.

Corollary 1 predicts that as code architecture becomes increasing more modular, the percentage of contributors out of all the enlisted programmers will increase first and then decreases afterward. We test

this prediction and show the results in Table 4. As shown in both Model 1 and Model 2, the coefficient on *Module* is positive and significant and that on *Module*^{2} is negative and significant, thus Corollary 1 is supported.

Table 4. Estimation Results for Corollary 1

Independent Variables	Model 1	Model 2
Project size	−0.032*** (0.004)	−0.031*** (0.004)
Module	0.207*** (0.057)	0.201*** (0.060)
Module^2	−0.057** (0.029)	−0.053** (0.031)
R^2	0.295	0.344

Notes: N=517. Dependent variable is *Pert*, the percentage of positive contributors among all programmers within a project. **p<0.05, ***p<0.01. Model 2 includes 34 dummy variables as in Table 3.

Next we test how modular architecture will affect free-riding. Proposition 4 predicts that, overall, the extent of free-riding within a team is a concave function of total modules a project has. We test the following OLS model:

$$Gini = \gamma_1 Size + \gamma_2 Module + \gamma_3 Module^2 + \boldsymbol{\theta} Characteristics + \varepsilon \quad (12)$$

where *Gini* is calculated from Equation (7) based on commits and LOC respectively. We show the test results in Table 5. Model 1 and Model 2 are based on commits and Model 3 and Model 4 are on LOC made by programmers. The results show that coefficient on *module* is positive and significant while that on *Module*^{2} is negative and significant, thus confirming Proposition 4.

Table 5. Estimation Results for Proposition 4

Independent Variables	Model 1	Model 2	Model 3	Model 4
Project size	0.030*** (0.004)	0.032*** (0.004)	0.033*** (0.004)	0.034*** (0.004)
Module	0.180*** (0.050)	0.171*** (0.050)	0.162*** (0.047)	0.149*** (0.048)
Module^2	−0.061*** (0.026)	−0.058** (0.025)	−0.054*** (0.024)	−0.050** (0.024)
R^2	0.312	0.355	0.315	0.350

Notes: N=517. Dependent variable is Gini_commit in Model 1 and 2, and Gini_LOC in Model 3 and 4. ***p<0.01. Model 2 and 4 further include 34 dummy variables as in Table 4.

Based on the result in Model 4 in Table 5, we plot the impact of number of module on the value of *Gini* in Figure 2 as below. It clearly shows a concave relationship between the two values.

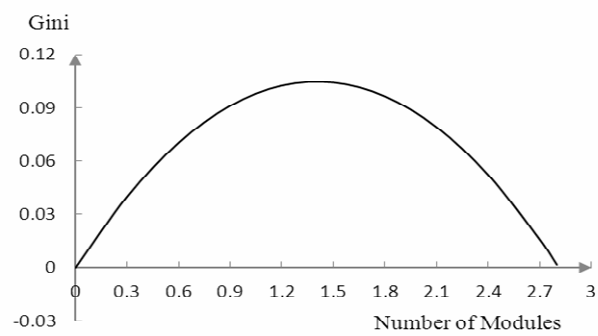

Figure 2. The Impact of Number of Modules and Gini

As discussed in Proposition 4, when the code architecture gets increasingly modular, hackers will reduce their contribution first, followed by the onlookers and finally the leaders. To validate this prediction, we run the following regression at programmer level rather than project level with four interaction terms added:

$$\begin{aligned} Contribution &= \alpha_1 Size + \alpha_2 Module + \alpha_3 Module^2 + \alpha_4 Hacker * Module + a_5 Hacker * Module^2 \\ &+ \alpha_6 Onlooker * Module + a_7 Onlooker * Module^2 + a_8 Hacker + a_9 Onlooker + \boldsymbol{\beta} Characteristics + \varepsilon \end{aligned} \tag{13}$$

The regression results for Equation (13) are shown in Table 6.

Table 6. Estimation Results for Equation (13)

Independent Variables	Model 1	Model 2
Module	2.601*** (0.413)	2.821*** (0.504)
Module²	−0.515** (0.222)	−0.585** (0.259)
Hacker*Module	3.618*** (0.904)	3.965*** (1.106)
Hacker*Module²	−1.914*** (0.562)	−2.127*** (0.680)
Onlooker*Module	0.555 (0.570)	0.287 (0.727)
Onlooker*Module²	−0.576** (0.291)	−0.486 (0.357)
R^2	0.264	0.225

Notes: N=1715. ***p<0.01. Dependent variable is the Code_commit in Model 1, and Code_LOC in Model 2. Other control variables are the same as in Equation (11).

It can be seen that compared with the base group of leaders, hackers increase their contribution faster initially, as evidenced by the positive and significant α_4 . However, their contribution decreases faster than others after certain threshold of *m*, and this is supported by the negative and significant sign of α_5 .

Discussion

We develop a model to examine the impact of modular code architecture on code contribution in OSS development. Our model explicit considers the resource heterogeneity, various motivations of the programmers as well as the strategic interactions between them, thus extending the results from prior literature (e.g., Johnson 2002; Baldwin and Clark 2006; Shah 2006). We make several important contributions to the emerging literature of OSS development.

First, current OSS models almost invariantly consider OSS as a pure public good since the final software products are shared by everybody, and assume contributors do not derive private benefits from their own contributions (e.g., Johnson 2002; Baldwin and Clark 2006). However, as described in the private-collective innovation model, motivations of various types of contributors to OSS projects are very different, and while some mainly benefit from the final product, others may derive benefits from the coding process itself (von Hippel and von Krogh 2003). Essential to our analysis is the presence of hacker programmers, which contribute to the source code most out of their own private interests (Shah 2006). To the best of our knowledge, our type-dependent model is the first to explicitly account for both the public and the private nature of the OSS model. Second, while prior literature have suggested that code architecture will influence the extent of free-riding which endangers the healthy development of OSS communities, they do not distinguish between code contribution and code participation (e.g., Johnson 2002; Baldwin and Clark 2006). We extend prior literature by arguing that these are two different concepts depicting involvement into OSS development: while two programmers may both participated in the coding activities, the amount of the code they contributed can differ substantially, thus describing free-riding as coding the OSS project or not is not accurate, and may even be misleading.

Third, our results reveal that rather than uniformly reducing free-riding in OSS development, modular architecture may well increase free-riding initially. It is due to tradeoff between the positive and negative impact of modular design. When modular architecture is initially adopted, the benefits outweigh the costs for programmers, particularly for the hackers, and this has widened the gap of code contributions between programmers. However, as code architecture gets increasingly modular, the negative side dominates, and programmers, particularly the hackers will refrain from contributing, which narrows down the gap of contributions, thus free-riding is less severe. Fourth, although theoretical models examining the effect of code architecture on code contribution exist, we know no empirical evidence in the current literature. In this study, we use the OSS development data obtained from SofrceForge.net. The CVS log file allows us to trace which developers contributed what codes to which modules and when. Therefore, our study also represents the first attempt to validate the theoretical models using real world data.

Our results contribute to our understanding about the broader literature of open innovation as well. Open innovation is attracting increasing attention from both researchers and practitioners, and it reflects the innovation trend that firms and organizations are increasing rely on the external as well as the internal resource for innovation due to stronger global competition and accelerated flow and exchange of knowledge and information across firm boundaries (Chesbrough 2003). The idea of open innovation originated from OSS development (e.g., Gruber and Henkel 2006, West and Gallagher 2006), and has quickly proliferated into other research areas as well. Researchers suggest that one of the most important challenges to open innovation is how to design the model so to facilitate innovation processes (Chesbrough 2007). So far, quite some research has devoted to the study of organizational design (e.g., Jacobides and Billinger 2006; Dittrich and Duysters 2007), but few have realized that product design also bears importance to the success of the open innovation. Our results shows both the positive and negative side of modular product design, thus managers can leverage the findings of this study so to enhance the contribution from various business partners, and eventually leading to success of open innovation.

Our results bear important implications for practitioners as well. First, to induce code contribution, it is imperative to attract the hacker programmers. Research has shown that hackers tend to be highly skilled,

and they can potentially provide guidance to other programmers (Shah 2006). Most importantly, they contribute to the code base out of their own private interests, thus their enthusiasm provides the bottom line to the healthy development of the OSS community. In contrast, the onlookers and the leaders anticipate the contribution levels from the hackers, and the less contribution from the hackers, the more likely the onlookers and the leaders tend to start to contribute. Second, it would help to make the code structure or the workspace more user-friendly, so that potential programmers can make up their mind to contribute, and a modular code structure is one of the effective approaches to do so. Third, and most importantly, both our theoretical and empirical analysis suggest that although modular design can potentially increase the level of contribution, it might not be reliable to count on the design structure to reduce free-riding or inequality of contribution. Thus practitioners need to avoid the negative impact of modular design and incorporate other mechanisms that potentially can help to solve the free-riding problem.

Conclusion

The private-collective nature of OSS model makes many believe that OSS model has the potential to be applied in much broader fields other than software development (von Hippel and von Krogh 2003; Rai 2005; Shah 2006; Fleming and Waguespack 2007). However, OSS model differs from prior models of innovations in some prominent ways, and success of OSS model depends on our understanding of some important issues such as the motivations of contribution and design of code architecture, etc. In this paper, we study how OSS code architecture could affect code contribution from programmers, with particular attention to the problem of free-riding. Our study opens several areas for future explorations. First, we do not differentiate the three types of benefits brought by the modular design: reduced cognitive cost, lower coordination cost, and enhanced code quality, future study might model and empirically test these effects separately. Another limitation is that we cannot distinguish all types of programmers directly from our dataset, and future studies might develop algorithms to identify programmer types so to validate our prediction for each type directly. Lastly, we model the OSS code contribution as a static game; however, code contribution involves constant interactions between various participants, thus future effort can extend this research by studying the dynamics of code contribution using dynamic model of multiple stages or even under continuous time horizon.

References

Bagozzi, R. P., U. M. Dholakia. "Open source software user communities: A study of participation in Linux user groups," *Management Science* (52:7), 2006, pp. 1099–1115.

Baldwin, C. Y., K. B. Clark. "The architecture of participation: Does code architecture mitigate free riding in the open source development model?" *Management Science* (52:7), 2006, pp. 1116–1127.

Bentley, J. "Multidimensional divide-and-conquer," *Communications of the ACM*. (23:4), 1980, pp. 214–229.

Chesbrough, H. W. *Open Innovation: The New Imperative for Creating and Profiting From Technology*. Harvard Business School Press, Boston, MA, 2003.

Chesbrough, H. W. "Why companies should have open business models" *MIT Sloan Management Review* (48:2), 2007, pp. 22–28.

Cooper, R. G. *Winning at New Products: Accelerating the Process from Idea to Launch*, Addison-Wesley, Reading, MA, 1993.

Dam, K. W. "Some economic considerations in the intellectual property protection of software," *Journal of Legal Studies* (24:2), 1995, pp. 321–377.

Dittrich, K., G. Duysters. "Networking as a means to strategy change: The case of open innovation in mobile telephony," *Journal of Product Innovation Management* (24:5), 2007, pp. 510–521.

Dougherty, D., C. Hardy. "Sustained product innovation in large, mature organizations: overcoming innovation-to-organization problems," *Academy of Management Journal* 39(5), 1996, pp. 1120–

1153.
Fischbacher, U., S. Gachter. "Social preferences, belief, and the dynamics of free riding in public goods experiments," *American Economic Review* (100:1), 2010, pp.541-556.
Fitzgerald, B. "A critical look at open source," *Computer* (37:7), 2004, pp. 92–94.
Fixson, S. K. "Modularity and commonality research: Past developments and future opportunities," *Concurrent Engineering* (15:2), 2007, pp. 85–111.
Fleming, L., D. M. Waguespack. "Brokerage, boundary spanning, and leadership in open innovation communities," *Organization Science* (18:2), 2007, pp.165–180.
Fogel, K. *Producing Open Source Software*, O'Reilly Media, Inc, Cambridge, MA, 2006.
Gershenson, J. K., G. J. Prasad, Y. Zhang. "Product modularity: Definitions and benefits," *Journal of Engineering Design* (14:3), 2003, pp.295–313.
Grewal, R., G. L. Lilien, G. Mallapragada. "Location, location, location: How network embeddedness affects project success in open source systems," *Management Science* (52:7), 2006, pp.1043–1056.
Gruber, M., J. Henkel. "New ventures based on open innovation–an empirical analysis of start-up firms in embedded Linux," *International Journal of Technology Management* (33:4), 2006, pp.356–372.
Haefliger S., G. von Krogh, S, Spaeth. "Code reuse in open source software," *Management Science* (54:1), 2008, pp.180–193.
Hars, A., S. Ou. "Working for free? Motivations for participating in open-source projects," *International Journal of Electronic Commerce* (6:3), 2002, pp.25–39.
Henderson, R. M., K. B. Clark. "Generational innovation: the reconfiguration of existing systems and the failure of established firms," *Administrative Science Quarterly* (35), 1990, pp. 9–30.
Hertel, G., S. Niedner, S. Herrmann. "Motivation of software developers in open source projects: An internet-based survey of contributions to the Linux kernel," *Research Policy* (32), 2003, pp.1159–1177.
Jacobides, M. G., S. Billinger. "Designing the boundaries of the firm: From 'make, buy, or ally' to the dynamic benefits of vertical architecture," *Organization Science* (17:2), 2006, pp. 249–261.
Johnson, J. P. "Open source software: Private provision of a public good," *Journal of Economics & Management Strategy* (11:4), 2002, pp.637–662.
Kuk, G. "Strategic interaction and knowledge sharing in the KDE developer mailing list," *Management Science* (52:7), 2006, pp.1031–1042.
Lakhani, K. R., E. von Hippel. "How open source software works: "Free" user-to-user assistance," *Research Policy* (32:6), 2003, pp.923–943.
Langlois, R. N. "Modularity in technology and organization," *Journal of Economic Behavior & Organization* (49:1), 2002, pp.19–37.
Lenat, D. B. "Cyc: A large-scale investment in knowledge infrastructure," *Communications of the ACM* (38:11), 1995, pp.32–38.
Lerner, J., J. Tirole. "Some simple economics of open source," *Journal of Industrial Economics* (2), 2002, pp. 197–234.
Liebeskind J. P. "Knowledge, strategy, and the theory of the firm," *Strategic Management Journal* (17), 1996, pp. 93–107.
MacCormack, A., J. Rusnak, C. Y. Baldwin. "Exploring the structure of complex software design: An empirical study of open source and proprietary code," *Management Science* (52:7), 2006, pp.1015–1030.
Marwell, G., P. Oliver. *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge University Press, Cambridge, U.K, 1993.
Mockus, A., R. T. Fielding, J. D. Herbsleb. "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology* (11:3), 2002, pp.1–38.
Osterloh, M., S. Rota. "Open source software development-Just another case of collective invention?" *Research Policy* 36, 2007, pp.157–171.
Pavlicek, R. C. Embracing Insanity: Open Source Software Development, Sams, Indianapolis, IN, 2000.
Parnas, D. L. "On the criteria to be used in decomposing systems into modules," *Communications of the ACM* (15:12), 1972, pp.1053–1058.
Penrose, E. T. *The Theory of the Growth of the Firm*, Oxford University Press, NY, 1995.
Rabin, M. "Incorporating fairness into game-theory and economics," *American Economic Review* (83:5), 1993, pp.1281–1302.

Rai, A. "Open and collaborative research: A new model for biomedicine," in *Intellectual Property Rights in Frontier Industries: Software and Biotechnology*, edited by R. Hahn, AEI-Brooking Joint Center for Regulatory Studies, Washington, D.C., 2005.

Raub, W. "Problematic social situations and the large number dilemma: A game theoretical analysis," *Journal of Mathematical Sociology* 13, 1988, pp.311–357.

Raymond, E. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly & Associates, Sebastopol, CA, 1999.

Roberts, J. A., I. Hann, S. A. Slaughter. "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects," *Management Science* (52:7), 2006, pp.984–999.

Schilling, M. A. "Toward a general modular systems theory and its application to interfirm product modularity," *Academy of Management Review* (25:2), 2000, pp. 312–334.

Sen, A. *On Economic Inequality*, Oxford University Press, 1973.

Shah, S. K. "Motivation, governance, and the viability of hybrid forms in open source software development," *Management Science* (52:7), 2006, pp.1000–1014.

Stewart, D. "Social status in an open-source community," *American Sociological Review* (70), 2005, pp. 823–842.

von Hippel, E. "Task partitioning: An innovation processing variable," *Research Policy* (19:5), 1990, pp. 407–418.

von Hippel, E., G. von Krogh. "Open source software and the "private-collective" innovation model: Issues for organization science organization science," *Organization Science* (14:2), 2003, pp.209–223.

von Krogh, G., S. Spaeth. "The open source software phenomenon: Characteristics that promote research," *Journal of Strategic Information Systems* (16), 2007, pp. 236–253.

von Krogh, G., E. von Hippel. "The promise of research on open source software," *Management Science* (52:7), 2006, pp.975–983.

West, J., S. Gallagher. "Challenges of open innovation: The paradox of firm investment in open-source software," *R&D Management* (36:3), 2006, pp. 319–331.