2012 45th Hawaii International Conference on System Sciences

Modularity and Inequality of Code Contribution in Open Source Software Development

Gang Peng
Williamson College of Business
Youngstown State University
gpeng@ysu.edu

Xianjun Geng
School of Management
University of Texas at Dallas
geng@utdallas.edu

Lihui Lin
School of Economics and Management
Tsinghua University
lin.lihui@gmail.com

Abstract

We develop a model to study how the code architecture affects open source software (OSS) development. A major advantage brought by a modular architecture of code base is that it reduces both the cognitive cost and the coordination cost associated with OSS development. We show that in OSS development, the amount of individual code contribution, the inequality of code contribution among programmers, and the total output of code all have nonlinear relationships with the level of modularity of the project. We further empirically test the predictions using the SourceForge OSS development data, and the results confirm our theoretical predictions.

1. Introduction

Open source software (OSS) has attracted researchers' attention largely because of its unique approach of software production [31, 32, 49]. With the success of open source projects such as Linux, Apache, MySQL, Perl, PHP, and Mozilla, the OSS development model not only has changed the landscape of the software industry, but also has important implications for product development and innovation in general [15, 40, 45, 49].

One of the key questions in OSS research is the motivations for programmers to contribute to the code base. Under the traditional software development model, programmers are hired and collocated to code the software projects. But in the open source model, programmers are not typically paid and the software is available to anybody for free [42], thus why programmers are willing to participate in the development of OSS has puzzled many researchers.[1] Recent studies have revealed that programmers do not always contribute for free; rather, they contribute out of complex and intertwined motivations, and they recoup certain private benefits while contributing to OSS projects [3, 31, 32, 43]. Consequently, researchers have proposed that the OSS model is a private-collective innovation model and represents a rich and fertile middle ground where incentives for private investment and collective action can coexist [20, 49].

A related problem is free-riding or inequality of code contribution to OSS development: although many projects can potentially attract a large number of programmers, quite often only a small number of individuals code the software [13, 27]. If not properly addressed, the free-riding problem can potentially derail the development of OSS projects [4, 42], 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.

Recent studies propose that modular code architecture can potentially reduce free-riding in OSS development [4, 25]. In a modular design, the whole software code base for one project is divided into several loosely coupled segments or modules that communicate with each other through common interfaces [38], and consequently the code structure is transformed from a monolithic architecture into a modular architecture. The responsibility for developing each module can be delegated to specific individuals [36], and subsequently participation tends to be equalized under the modular architecture [4, 25]. However, as far as we know, no research has examined the effect of modular architecture on OSS development.

Our research contributes to the literature in several important ways. First, OSS is typically considered as a pure public good in prior studies [4, 25], while our model incorporates both the public goods characteristics and the private aspects of OSS development. Second, in the literature code contribution by participants is often modeled as a binary variable (programmers either code or do not

[1] Although the popular image of the OSS movement portends an entirely volunteer enterprise, the possibility of paid participation should not be ignored as an obvious first-order explanation of extrinsic motivations.

978-0-7695-4525-7/12 $26.00 © 2012 IEEE
DOI 10.1109/HICSS.2012.437

code), but we consider the differences in the level of contribution by participants. Third, in addition to the diverse motivations, we also consider the resource endowments of the participants, which prove to play critical role in their reaction to the design of code structure. Fourth, our research is the first attempt to test the effect of modular architecture on OSS development empirically.

2. Literature Review

Innovations are critical for the competitiveness of firms and nations [8, 12, 39]. Traditionally, two models of innovation development have been practiced: the private model and the collective model. In the private model, innovators invest private resources to develop the innovation, and in return, they own the innovation and collect private returns typically through property rights [2, 9]. In the collective model, practiced mostly in science, the innovators usually are subsidized to develop the innovation and they relinquish control of the knowledge developed and make it a public good by unconditionally supplying the innovation to a "common pool" [33, 37]. However, the OSS model deviates from both these two models of innovation in that innovators invest their own private resource for the development of a public good, thus the OSS model potentially represents a new type of innovation [49]. Contributions to OSS development are not pure public goods — innovators have significant private considerations even though the innovation will be freely revealed, thus open source has been termed as a private-collective model of innovation [49].

The private aspect of OSS contribution can be clearly seen from the diverse motivations of programmers. 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" [42, 49], affiliation and identity [23], signaling and career concern [31], peer recognition [22, 28], reputation and status [43], and hedonic motives such as enjoying programming [23, 45], 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 [31]. Second, the various benefits can fall into two categories: private benefits such as fun, knowledge, and career concern, and public benefits or the delivery of the final OSS products. Third, whether participants derive private benefits or public benefits depends critically on the roles they play during the OSS development process.

Prior literature suggests that participants in the OSS community mainly belong to two types, the hobbyists who derive benefits from their own coding activities and the need-driven participants who need to use the OSS for their own purposes [45]. Hobbyists are also referred as hackers in literature [49]. By contributing to OSS code, hackers derive fun, knowledge from their contribution, and gain peer recognition and reputation [42, 46], and even signal their talents to attract prospective employers and venture capital [31]. In communities of open source programmers, "hacker" is a very positive term that is applied to very talented and dedicated programmers [49], and hackers give and exchange their software so others can modify and build upon so that the community can form a "hacker culture" in which software is freely exchanged and developed both individually and collaboratively.

A closer examination of the roles played by 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 [49], and they set up the virtual workspaces and provide the initial code base for the projects [45]. The primary motivation for leaders is the delivery of the final product [42, 49]. 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. They tend to contribute when they have to (e.g., when the OSS project is in slow progress or risk abandonment due to lack of contribution), which is in clear contrast to hobbyists who contribute when they want to.

One of the key design rules in software development is modularity. Modularity is a general systems concept [47] that describes the degree to which a system's components can be separated and recombined (Schilling 2000). In a complex system made up of large number of components that interact with each other, a modular design can group the components into a smaller number of subsystems so as to reduce the interdependency between each component [29]. Software development is one of the areas that have witnessed the most mature application of modular design of products [14]. Parnas [38] argues that a software module captures a set of design decisions which are hidden from other modules, and the interactions among the modules should primarily be through module interfaces, and that modular design promotes encapsulation or information hiding by separating the module's interface from its implementation. The benefits of modular architecture include concurrent development, robustness to interruption of the production process, reduced

communication cost, recombination of source code, code reuse, and increased quality, etc. [17, 20]. 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 regard, two most important studies are [4, 25].

Johnson [25] models OSS development as a static Bayesian game of code contribution. In his model, a programmer can either choose to code one and only one module under the modular architecture, or code the whole project under the monolithic architecture if (s)he chooses to code. Johnson finds that whether modular design will encourage contribution depends on the size of the project: when the number of programmers is large enough, modular design out-performs monolithic design; the situation reverses when there are few programmers. In other word, modular design has the potential to induce contribution for larger projects.

Extending Johnson's work, Baldwin and Clark [4] model OSS development as a game of complete and perfect information. In their study, OSS code base consists of multiple modules and each programmer chooses to work on one and only one module. In other words, each programmer faces a binary choice, either to code or not to code. They show that under modular design, the code contribution game has multiple equilibria, and programmers assign themselves to corresponding modules so that the workload is evenly distributed among programmers. In other words, modular code architecture can significantly reduce free riding.

While path-breaking and insightful, the works of Johnson [25] and Baldwin and Clark [4] leave some questions unanswered. We believe that some of the important aspects of OSS development need to be incorporated when studying code contribution: 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 [42]. Second, code contribution is a collective effort, thus OSS code contribution model should be flexible enough to accommodate concurrent contributions and different levels of contributions, instead of assuming that a single module is coded only by one programmer at one point in time. Indeed, early observations of the process of OSS development reveal that OSS workspace is very much like a "bazaar," where programmers contribute to the source code collectively [42]. Third, we believe that it is important to distinguish between two concepts in studying OSS development: code participation and code contribution. Code participation is a binary variable: either to code the project or not [4]; in contrast, code contribution is a continuous variable that measures how much a programmer codes in terms of lines of code (LOC) or commits made to the code base. In other words, code contribution is a more accurate measure of free-riding than code participation.

Next we develop a model for OSS code contribution that incorporates the above features and analyze the effect of code architecture on code contribution.

3. Model

Programmers working on an OSS project incur a variety of costs. By engaging in an OSS project, programmers devote their time that otherwise could be used working on other programming tasks, such as their own proprietary projects [31, 45]. Suppose there is one OSS project which enrolls N programmers including leaders, onlookers, and hobbyists [49]. Beside the OSS project, programmers also engage in their own proprietary projects. A programmer i is endowed with total available work time of w_i. A programmer can allocate $0 \le x_i \le w_i$ units of time to their proprietary projects. A programmer can spend the rest of his time on an OSS project and his utility from this activity depends on the modularity of the project and his characteristics as a developer.

In OSS development, modular code architecture can potentially affect the code contribution through various mechanisms. First, a modular structure reduces the cognitive cost of the potential contributors. Modular design divides the project into loosely coupled components, which are less complex [48] and can be more easily identified and managed [1]. Thus smaller and less complex code components decrease the amount of information needs to be processed. Indeed, modular design is consistent with the "divide-and-conquer" philosophy in software engineering [5, 30]. Second, a modular structure reduces the coordination cost among the project participants. Due to the loosely-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 [34]. In other words, modular design supports "information hiding" which is one of the most important principles for modern programming. Therefore, a modular design significantly cuts down coordination costs. Third, because modules within the code base do not talk to each other directly, code modifications made within one module do not interfere with the functioning of other modules, and code produced using modular architecture tends to have fewer bugs and higher quality. Fourth, a modular architecture helps potential

contributors identify the code components that are most interesting to them, facilitating the code development. This is particularly important for OSS, because programmers usually assign themselves to the tasks rather than being assigned by the project leaders as in proprietary software development teams [45].

Back to our model, the time a programmer spends on an OSS project can be further divided into two parts: the time devoted to coding, defined as g_i, for the *i*th programmer, and the time spent on processing information about the module(s) he is working on and coordinating with other developers. Based on our previous analysis the second time component decreases when a project has a more modular structure. We use variable m to measure the level of modularity of the code structure: a higher m corresponds to a more modular design and vice versa. Define $C_i(g_i)$ as the coordination cost, then the time that programmer *i* spends on information processing and coordination on an OSS project can be written as C_i/m. It is understandable that the time for information processing and coordination is positively correlated with the time spent on coding: the more time a programmer devotes to coding (and thus generating more lines of code), the more time he has to allocate to coordinating with others. Therefore, the *i*th programmer's time constraint can be written as:

$$x_i + g_i + \frac{C_i(g_i)}{m} \le w_i$$

where $C_i(g_i)$ increases with g_i. Define $c_i(g_i) = dC_i/dg_i$. Thus $c_i(g_i) > 0$.

This time constraint shows that the variable m in fact determines the *effective* coding time that can be spent on the OSS project for any units of time taken away from developing the proprietary project. For any units of time taken away from developing the proprietary project, a higher m means more *effective* units of time can be devoted to the coding of the OSS project. This is precisely 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. Without loss of generality, the parameter m can be simply understood as the number of modules in a project. However, our model can also represent any alternative measure of modularity for code architecture.

The time spent on coding g_i produces $o_i(g_i)$ lines of code and the total lines 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. To simplify the analysis, assume the code production function for all programmers is the same, $o_i(g_i) = o(g_i)$, and $o(y_i)$ is linear and increasing in y_i. Therefore, we have: $O = \sum_N o(g_i) = o(\sum_N g_i) = o(G)$.

As discussed earlier, depending on their types, programmers potentially derive utilities from three sources: code contribution to their own proprietary projects by spending the time x_i; their private benefit from coding an OSS project for g_i amount of time; and the outcome of the whole OSS project, O. The utility function of programmer *i* can then be expressed as: $u_i(x_i, g_i, G)$. Thus a utility maximizing programmer *i* solves the following problem:

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

where G_{-i} is the total time devoted to coding by all other programmers except programmer *i*. We assume that u_i is increasing in all three variables.

As [4] and [25], the model retains the strategic interactions among contributors when deciding on their own contribution level. But our model has several advantages over models presented in prior studies. First, it considers code contribution as a continuous variable. Second, it allows multiple programmers to code the same module thus more closely reflects 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. The maximization problem shows that there is a tradeoff between the time programmer *i* allocates to his propriety project and to the OSS project.

Next we model the three types of programmers discussed earlier and study how a modular design

could potentially affect code contribution for different types of contributors.

As discussed earlier, leaders, onlookers, and hobbyists attach different importance to private benefit from coding and the whole OSS source code. First, prior literature has shown that project leaders attach more importance to the OSS project than the followers [32]. In contrast, hobbyists derive utilities mainly from their fun, knowledge, and reputation gained from coding their parts of OSS source code, as well as their own proprietary projects.

We use a parameter α_i to differentiate these three types of programmers in modeling their utilities from the OSS project. We assume that the utility function for code contribution to proprietary projects is the same for all three types, and is simply the time spent on proprietary projects, x_i. The model presented in (1) is thus transformed to:

$$\begin{aligned} \max_{x_i, g_i} \; & u_i = \alpha_i s(g_i, m) + (1-\alpha_i) r(G) + x_i \\ s.t. \quad & x_i + g_i + \frac{C_i(g_i)}{m} \le w_i \\ & G_{-i} + g_i = G \\ & g_i \ge 0 \\ & x_i \ge 0 \end{aligned} \tag{2}$$

where $\alpha_i s(g_i, m)$ is the utility function from private benefits, $(1-\alpha_i) r(G)$ is the utility function from public benefits, and $0 \le \alpha_i \le 1$. If α_i is low, it represents a leader; if α_i is high (but lower than 1), it represents an onlooker; if $\alpha_i = 1$, it means a hacker. Further, the following stylized properties are satisfied: $s_{g_i} > 0$, $s_m < 0$, $s_{g_i g_i} < 0$, $s_{g_i m} < 0$, $s_{g_i mm} < 0$, $s_{g_i m} < 0$, $s_{g_i g_i m} < 0$, $r_G > 0$, and $r_{GG} < 0$.

It is straightforward that the time constraint will be binding in equilibrium. Therefore, the maximization problem presented in (2) can be further simplified as:

$$\max_{g_i} \; u_i = \alpha_i s(g_i) + (1-\alpha_i) r(G_{-i} + g_i) + w_i - g_i - \frac{C_i(g_i)}{m} \tag{3}$$

We obtain the following propositions.

Proposition 1: For the code contribution game defined in the maximization problem (3), there exists a unique Nash Equilibrium where the ith programmer spends g_i^* units of time on an OSS project.

Proposition 2: The time the ith programmer contributed to coding an OSS project g_i^* is the highest for hackers, then onlookers, and the lowest for leaders.

Proposition 2 shows an interesting result: the ones with the most to gain contribute the least. Anticipating the contributions from hackers and onlookers, the leaders strategically hold back their efforts.

Proposition 3: There exists a unique level of modularity m_i^* for the ith programmer, at which his time coding the OSS project is maximized. Further, $\frac{dg_i^*}{dm} > 0$ if $m < m_i^*$, and $\frac{dg_i^*}{dm} < 0$ if $m > m_i^*$.

Corollary 1: For a given pool of N programmers with the type of each fixed, there exists a unique level of modularity m^* such that the total time contributed to an OSS project G is maximized.

Propositions 2 and 3, and Corollary 1, reveal the effects of modular architecture on the actual levels of code contribution for the OSS projects.

Next we model the inequality of code contribution among programmers. Note that inequality of code contribution is not merely free riding in code participation. In prior studies [4, 25], the term free riding is often used to describe the phenomenon where some programmers do not contribute at all and hence free ride over those who do contribute. In our model the level of contribution by each programmer is a continuous decision rather than a binary one, and some participants contribute more than others. Therefore, our discussion on inequality of code contribution is richer than the free riding issue studied in the literature - free riding would be an extreme case of our model where some participants choose not to contribute (i.e., $g_i^* = 0$ for some programmers).

There are various measures for inequality of resources or contributions in literature, and probably the best known is the *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 [44]. Consistent with prior studies on OSS code contribution [27], we use *Gini* coefficient to measure the inequality of code contribution among programmers within a project:

$$Gini = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \left| g_i - g_j \right|}{2N^2 \bar{g}} \tag{4}$$

where g_i (or g_j) is the contribution made by programmer *i* (or *j*), $\bar{g}$ the average contribution made by a whole project, and N is the number of programmers on the project (no matter they contribute

or not). In this study, we use two measures of code contribution to calculate the *Gini* coefficient: the number of commits and the lines of code (LOC) contributed by a programmer. It can be shown that $Gini \in [0,1]$, and the higher the value of *Gini*, the more severe the contribution inequality in the group [44]. 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 will affect contribution inequality:

Proposition 4: Let Gini coefficient be the measurement of contribution inequality, then $\frac{dGini}{dm} \geq 0$ *if* $m < m^*$ *and* $\frac{dGini}{dm} < 0$ *if* $m \geq m^*$.

Contrary to what was believed that a modular architecture will reduce inequality in code contribution, Proposition 4 shows that, when incorporating heterogeneity among participants, modular architecture could potentially increase contribution inequality due to the strategic interactions among the programmers.

4. Empirical evidence

To test the effect of code structure on OSS development, we use the SourceForge.net data dumps maintained by University of Notre Dame and Universidad Rey Juan Carlos of Spain. SourceForge.net is the world's largest OSS project-hosting website. Project leaders recruit other members and grant them access to code base so they can contribute source code to the projects. The data source hosted by University of Notre Dame University has information about the software project data such as project characteristics, developer characteristics, developer roles, and activities at project forums and bug tracking service, etc.[2] It has been extensively used for OSS development research [18]. However, to study how code structure affects code contribution, we need to know individual contributions to the source code by each developer, and to that end, we make use of the second dataset, the SourceForge OSS projects' CVS log file host by LibreSoft group at Universidad Rey Juan Carlos in Spain.[3] At SourceForge, programmers make changes to the source code through the Concurrent Versioning Systems or CVS [16]. In CVS, each change made to the source code is called a commit, and is recorded in the CVS log file. Within each commit, the CVS also records the lines of code (LOC) that are changes. Thus the CVS log file provides rich information for studying software development activities, as it enables us to trace exactly who contribute what to which module of the project over time. The CVS log file has also been used in other studies to examine the coding dynamics of OSS development [26].

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. We further restrict our samples to the projects that have at least two programmers in order to study contribution inequality. To avoid left censoring problem, we further restrict our samples to 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, the age of the project (in months), 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 calculation of the Gini coefficients requires two pieces of information: the total number of programmers on a project and each programmer's code contribution. Although the CVS log file has detailed coding activities for all contributors, it does not record those programmers who do not contribute at all; thus Gini coefficients calculated purely based on the contributors recorded in the CVS log file will be biased. To solve the problem, we match the projects in CVS log file to those in SourceForge data dump using the unique Unix project names,[4] so we can identify not only those who contributed but also those who did not.

To examine the coding activities of programmers, we need to identify 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

[2] For more details: http://www.nd.edu/~oss/Data/data.html
[3] For more details: http://libresoft.es/Results/

[4] The two datasets use different identifiers (project IDs), but they share the same distinct Unix project name. Thus we match the two datasets using the Unix project name as the project identifier.

having an average score of 4 or higher as hackers if they are not leaders. The summary statistics of our data show that leaders contribute less than the onlookers who further contribute less than the hackers, thus we conclude that Proposition 2 is supported.

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:

$$Contribution = \alpha_1 Size + \alpha_2 Module + \alpha_3 Module^2 + \boldsymbol{\beta} Characteristics + \varepsilon \quad (5)$$

where *Contribution* is the total code contribution of a project team, α_i are scalars and β is a vector to be estimated, and ε is the error term. The results are shown in Table 1. Models 1 and 3 do not include the research variables *Module* and *Module*^{2}. However, the explanatory power increases significantly in Models 2 and 4 where *Module* and *Module*^{2} are included. Moreover, in both Models 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 1. Estimation Results for Proposition 3

Independent Variables	Model 1	Model 2	Model 3	Model 4
Project size	0.262*** (0.024)	0.100*** (0.020)	0.269*** (0.028)	0.106*** (0.027)
Module		4.913*** (0.279)		4.920*** (0.376)
Module^2		−1.392*** (0.123)		−1.384*** (0.165)
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 Models 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 stage, intended audience, operating systems, and project topics.

Next we test how modular architecture will affect contribution inequality. We show the test results in Table 2, and the dependent variable is the Gini coefficient calculated from Equation (4), and the control variables are the same as in Equation (5). Models 1 and 2 are based on commits and Models 3 and 4 on LOC made by programmers. The results from all the models show that coefficient on *Module* is positive and significant, while that on *Module*^{2} is negative and significant, implying that modularity first increases the contribution inequality and then start to reduce it after certain threshold.

The results in Table 2 reveal another interesting pattern: as the project size (total programmers on the project) increases, Gini coefficient increases; in other words, contribution inequality is more severe in larger groups than in smaller groups. As project increase in size, it becomes increasingly costly for each group member to monitor others behavior, and eventually the public good is under-provided.

Table 2. Estimation Results for Proposition 4

Independent Variables	Model 1	Model 2	Model 3	Model 4
Project size	0.030*** (0.003)	0.032*** (0.003)	0.033*** (0.003)	0.034*** (0.003)
Module	0.180*** (0.041)	0.171*** (0.043)	0.162*** (0.043)	0.149*** (0.045)
Module^2	−0.061*** (0.018)	−0.058*** (0.019)	−0.054*** (0.019)	−0.053*** (0.019)
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 1.

5. Discussion

The private-collective nature of OSS model makes many believe that OSS model has the potential to be applied to the development of other products than software [15, 40, 45, 49]. However, OSS model differs from prior models of innovations in some important ways, and success of the 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. 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 [4, 25, 45]. We show that in OSS development, onlookers do not have as high stakes as the leaders and also lack the enthusiasms of the hobbyists, thus they have a higher propensity to contribute less. More importantly, our results indicate that programmers react to the design of code architecture strategically and, as a result, modular architecture does not necessarily reduce contribution inequality and in fact may increase inequality.

We make several important contributions to the OSS literature. First, while researchers have recognized the private benefits of OSS participants [49], prior studies often treat OSS as a pure public good in formal models, where the contributors do not derive private benefits from their own contributions

(e.g., [4, 25]). Motivations of contributors involved in an OSS project vary significantly, and while some mainly benefit from the final product, others may derive benefits from the coding process itself [49]. Essential to our analysis is the presence of hobbyist programmers, who contribute to the source code out of their own private interests [45]. Many of these programmers have attracted enormous attention and have continued to gain tremendous business success [31]. To the best of our knowledge, our research is the first to explicitly account for both the public and the private nature in an analytical model of OSS development.

Second, prior studies have discussed free riding by some programmers which endangers the healthy development of OSS communities, they do not distinguish between code contribution and coding participation [4, 25]. We extend prior research by arguing that code contribution and coding participation are two different concepts: while two programmers may both participate in the coding activities, the amount of the code they contribute can differ substantially. We believe that by studying the contribution inequality and treating code contribution as a continuous variable, we can gain more insight about OSS development than focusing on free riding alone, where contribution is considered a binary decision.

Third, prior studies suggest that a more modular design reduces free riding, which implies less inequality of code contribution. Our theoretical results show that modular architecture may either increase or decrease inequality of code contribution, and our empirical results confirm that inequality may increase with modularity.

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 with developer contributed what codes to which modules. Therefore, our study also represents the first attempt to validate the theoretical findings using data.

Our results also contribute to the understanding about the management of open innovation in general. Open innovation is attracting more and more attention from both researchers and practitioners, and it reflects the trend that firms and organizations increasingly rely on external resources as well as internal resources for innovation due to global competition and accelerated flow and exchange of knowledge and information across firm boundaries [6]. The idea of open innovation has originated from OSS development [19, 50], and has quickly proliferated into other research areas as well. Researchers suggest that one of the most important challenges to open innovation is the design of business models so to facilitate the innovation process [7]. Many studies have been devoted to the study of organizational design [11, 24], but few have realized that product design also bears importance to the success of the open innovation. Our results show that modular product design can potentially enhance the total contribution from various business partners, thus leading to success of open innovation.

This study also adds to our understanding of public goods in general. Provision of private rewards for contributions to a public good has very significant implications for the governance of collective action projects. A major concern for governance of collective action projects is how to discourage free-riding. Conventional wisdom suggests that in larger group, collective action becomes increasingly fragile [41], since social relationships become increasingly scattered and ephemeral, and interests becomes increasingly diverse. When the group increases in size, the impact of any individual's participation in producing the collective good is negligible, and a self-interested, rational individual will choose not to contribute under these conditions [21]; an individual's decision to shirk is spread over a greater number of people [10], and the cost of organizing and inducing cooperation increases as well [35]. Since OSS development bears the public good nature, this is validated in our empirical results as well.

Our results bear important implications for practitioners as well. First, to induce code contribution, it is imperative to attract the hobbyist programmers. Research has shown that hobbyists tend to be highly skilled, and they can potentially provide guidance to other programmers [45]. 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 hobbyists. The less the stock of the code base available from the hobbyists, the more likely they 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 be a delusion to count on the design structure to reduce inequality of contribution. Thus practitioners need to explore other mechanisms that potentially can promote equal contribution. In fact, as suggested by our model,

several possible ways exist that can induce code contribution. For example, potential programmers can be introduced to the features and functions of the OSS project so that programmers can attach more importance to it and devote resource on it.

There are several areas for future explorations. First, we do not differentiate between the three types of benefits brought by the modular design: reduced cognitive cost, lower coordination cost, and enhanced code quality. Future studies may model and empirically study these effects separately. Another limitation is that we model the OSS code contribution as a simultaneous game. Code contribution involves constant interactions between participants, thus future effort can extend this research by studying the dynamics of code contribution in multiple stages.

References

[1] Alexander, C. 1964. *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, MA.

[2] Arrow, K. J. 1962. Economic welfare and the allocation of resources for invention. In The rate and direction of inventive activity: Economic and social factors (pp. 609-626). Princeton, NJ: Princeton University Press, National Bureau Economic Research.

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

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

[5] Bentley, J. 1980. Multidimensional divide-and-conquer. *Communications of the ACM*. 23(4) 214−229

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

[7] Chesbrough, H. W. 2007. Why companies should have open business models. *MIT Sloan Management Review* 48(2) 22−28

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

[9] Dam, K. W. 1995. Some economic considerations in the intellectual property protection of software. *Journal of Legal Studies* 24(2) 321−377.

[10] Dawes, R. 1980. Social dilemmas. *Annual Review of Psychology* 31 169−193.

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

[12] Dougherty, D., C. Hardy. 1996. Sustained product innovation in large, mature organizations: overcoming innovation-to-organization problems. *Academy of Management Journal* 39(5) 1120–1153

[13] Fitzgerald, B. 2004. A critical look at open source. *Computer* 37(7) 92−94.

[14] Fixson, S. K. 2007. Modularity and commonality research: Past developments and future opportunities. *Concurrent Engineering* 15(2) 85−111.

[15] Fleming, L., D. M. Waguespack. 2007. Brokerage, boundary spanning, and leadership in open innovation communities. *Organization Science* 18(2) 165−180.

[16] Fogel, K. 2006. *Producing Open Source Software*. Cambridge, MA: O'Reilly Media, Inc.

[17] Gershenson, J. K., G. J. Prasad, Y. Zhang. 2003. Product modularity: Definitions and benefits. *Journal of Engineering Design* 14(3) 295−313.

[18] Grewal, R., G. L. Lilien, G. Mallapragada. 2006. Location, location, location: How network embeddedness affects project success in open source systems. *Management Science* 52(7) 1043–1056.

[19] Gruber, M., J. Henkel. 2006. New ventures based on open innovation–an empirical analysis of start-up firms in embedded Linux. *International Journal of Technology Management* 33(4) 356−372.

[20] Haefliger S., G. von Krogh, S, Spaeth. 2008. Code reuse in open source software. *Management Science* 54(1) 180−193.

[21] Hardin, R. R. 1971. Collective action as an aggregate N-prisoner's dilemma. *Behavioral Sci*. 16 472–481.

[22] Hars, A., S. Ou. 2002. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce* 6(3) 25−39.

[23] Hertel, G., S. Niedner, S. Herrmann. 2003. Motivation of software developers in open source projects: An internet-based survey of contributions to the Linux kernel. *Research Policy* 32 1159−1177.

[24] Jacobides, M. G., S. Billinger. 2006. Designing the boundaries of the firm: From "make, buy, or ally" to the dynamic benefits of vertical architecture. *Organization Science* 17(2) 249−261.

[25] Johnson, J. P. 2002. Open source software: Private provision of a public good. *Journal of Economics & Management Strategy* 11(4) 637−662.

[26] Koch S., G. Schneider. 2000. Results from software engineering research into open source development projects using public data. Work paper, University of Economics and Business Administration, Vienna, Austria.

[27] Kuk, G. 2006. Strategic interaction and knowledge sharing in the KDE developer mailing list. *Management Science* 52(7) 1031−1042.

[28] Lakhani, K. R., E. von Hippel. 2003. How open source software works: "Free" user-to-user assistance. *Research Policy* 32(6) 923−943.

[29] Langlois, R. N. 2002. Modularity in technology and organization. *Journal of Economic Behavior & Organization* 49(1) 19−37.

[30] Lenat, D. B. 1995. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM* 38(11) 32–38.

[31] Lerner, J., J. Tirole. 2002. Some simple economics of open source. *Journal of Industrial Economics* 2 197−234.

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

[33] Liebeskind J. P. 1996. Knowledge, strategy, and the theory of the firm. *Strategic Management Journal* 17 93–107.

[34] MacCormack, A., J. Rusnak, C. Y. Baldwin. 2006. Exploring the structure of complex software design: An empirical study of open source and proprietary code. *Management Science* 52(7) 1015−1030.

[35] Marwell, G., P. Oliver. 1993. *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge University Press, Cambridge, U.K.

[36] Mockus, A., R. T. Fielding, J. D. Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11(3) 1−38

[37] Osterloh, M., S. Rota. 2007. Open source software development-Just another case of collective invention? *Research Policy* , 36, 157−171.

[38] Parnas, D. L. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 15(12) 1053−1058.

[39] Penrose, E. T. 1995. *The Theory of the Growth of the Firm*. Oxford University Press, NY

[40] Rai, A. 2005. 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.

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

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

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

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

[45] Shah, S. K. 2006. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science* 52(7) 1000−1014.

[46] Stewart, D. 2005. Social status in an open-source community. *American Sociological Review* 70 823−842.

[47] Ulrich, K. 1995. The role of product architecture in the manufacturing firm. *Research Policy*. 24(3) 419−440.

[48] von Hippel, E. 1990. Task partitioning: An innovation processing variable. *Research Policy* 19(5) 407−418.

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

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