MIS Quarterly

RESEARCH ARTICLE

FOLLOWERSHIP IN AN OPEN-SOURCE SOFTWARE PROJECT AND ITS SIGNIFICANCE IN CODE REUSE[1]

Qiqi Jiang
Department of Digitalization, Copenhagen Business School, Howitzvej 60,
Frederiksberg 2000 DENMARK {qj.digi@cbs.dk}

Chuan-Hoo Tan
Department of Information Systems and Analytics, National University of Singapore,
13 Computing Drive, 117417 SINGAPORE {tancho@comp.nus.edu.sg}

Choon Ling Sia
Department of Information Systems, City University of Hong Kong, Tat Chee Avenue 83, Hong Kong, SAR CHINA {iscl@cityu.edu.hk}
and Department of Information Management, National Taiwan University, Taipei City 106, Taiwan R.O.C. {iscl@cityu.edu.hk}

Kwok-Kee Wei
Department of Information Systems and Analytics, National University of Singapore,
13 Computing Drive, 117417 SINGAPORE {weikk@comp.nus.edu.sg}

Code reuse is fundamental to the development of open-source software (OSS). Therefore, understanding how and why it occurs is important. To date, researchers have examined code reuse in OSS largely from the perspective of leaders. We show why followers must be considered as well. "Followers" are people who have had previous contacts with an individual from another project and who continue to associate with him or her. We consider two types of followers: developers (those directly involved in software development) and observers (those indirectly involved in it). We conduct a series of empirical investigations by using a longitudinal dataset of OSS projects hosted in GitHub, along with a survey and qualitative data. We find that followership can affect code reuse, but the effect depends on the nature of the follower (developer or observer). Overall, our study suggests that followership is important for code reuse in OSS because it enables participants to learn, and learning promotes code reuse.

Keywords: Open-source software, code reuse, developer, observer, followership theory

Introduction

> *Contributing to open-source software is a profoundly social activity…virtually all new open-source projects derive from those [other OSS] projects and the developers that preceded them, creating a vast body of work that accelerates innovation and fuels further collaboration.*
>
> Tim Yeaton,[2] CEO, Black Duck

As quoted above, open-source software (OSS) is often created partly through a practice known as code reuse, which recom-

[1]Andrew Burton-Jones was the accepting senior editor for this paper. Kathy Chudoba served as the associate editor.

The appendices for this paper are located in the "Online Supplements" section of *MIS Quarterly*'s website (https://misq.org).

[2]Source: http://mashable.com/2011/09/28/open-source-social/ (last accessed October 23, 2017).

bines architecture, features, and code segments from other OSS projects (Baldwin and Clark 2006; MacCormack et al. 2006). An example of OSS is the Debian project, whose code was reused by other projects (e.g., Ubuntu) more than 40 times in 2012 alone. Engaging in reuse offers tangible benefits in OSS (as reuse improves coding efficiency and quality) and intangible benefits (as reuse signals a project's contribution to other communities) (Crowston et al. 2003; Haefliger et al. 2008; von Krogh et al. 2005). Engaging in reuse also reinforces the spirit of OSS: "a worldwide community of ethical programmers dedicated to the cause of [software] freedom and sharing" (Kovarik 2015, p. 318).

Despite the importance of code reuse in OSS, our understanding of it is largely restricted to its facilitation of software innovation and knowledge transfer (Haefliger et al. 2008; Rus and Lindvall 2002; Spinellis and Szyperski 2004). We still know little of how and why code reuse occurs (von Krogh et al. 2003). A plausible way to address this issue is to consider the participants involved. For example, the Ubuntu project's reuse of code from the Debian project is not surprising given that nearly 45% of Ubuntu's core development team members are also affiliated with the Debian project.[3]

This research contributes to understanding the role of participants in code reuse by considering the distinction between leaders and followers. Past OSS research has primarily examined leaders,[4] but we examine the role of followers. We anchor our work in followership theory (Howell and Shamir 2005; Uhl-Bien et al. 2014) and distinguish between two types of followers: developers and observers. In the context of OSS, "followers" refers to people who have had previous contacts with an individual from another project and who continue to associate with (or follow) him or her. We propose that followership affects code reuse, but the effect can vary depending on the nature of the follower (developer or observer). Specifically, we postulate that having diverse developers following a leader can foster reuse in other projects through their direct involvement in these other projects, whereas having a large number of observers following a leader can foster reuse through observers' sharing and communication interactions in other projects. The effects are different, but they share a common underlying mechanism for code reuse, that is, reuse enables learning. We substantiate our thesis via two empirical means. First, we statistically validate our conjectures by using an OSS dataset collected over two consecutive periods in 2012 and 2014 through the GitHub platform. Next, we conduct a survey and qualitative interviews with participants from the same platform.

Our focus on followers rather than leaders in this study fills an important void in the literature. Although we need to understand the influence of leaders, followers comprise a much larger proportion of a project's participants. This study provides a first step in understanding the effects of followers. Our research also raises interesting questions about followership that offer opportunities to both understand OSS more deeply, and to refine followership theory.

Background Concepts: Followership, OSS Roles, and Learning

Followership and OSS Roles

Followership theory, which is an extension of leadership theory (Howell and Shamir 2005; Uhl-Bien et al. 2014), dictates the inclination and action of an individual to follow another individual (i.e., a leader) to jointly accomplish common tasks (Riggio et al. 2008). An often cited reason for focusing on followers rather than a leader is that leadership only occurs when followers exist. The rationale for studying followers rather than identifying the traits and styles of leaders, although somewhat simplified, is the belief in people's autonomous choice of the following behavior, such as in the case of OSS (Bono and Judge 2004).

Further clarity is added when conceptualizing followership through the role-based perspective (Collinson 2006; Lord and Brown 2003). These studies have focused on characterizing and demarcating the role of followers by unveiling how followers function under leaders (Collinson 2006; Lord and Brown 2003). In this study, we build on the role-based perspective to study the effect of the different roles of followers on code reuse in OSS projects.

Previous literature has categorized the people involved in an OSS project into successive layers of roles (Aberdour 2007; Crowston and Howison 2006; Setia et al. 2012). Although the terminology used for describing such roles varies, substantial commonality among these studies is evident. The first commonality is that these studies have defined roles on the basis of the extent of OSS participants' engagement in the software development. For example, Crowston and Howison (2006)

[3]Source: https://launchpad.net/~ubuntu-core-dev/+members#active (last access October 23, 2017). A total of 83 direct members comprise the "Ubuntu Core Development Team" (one member named "Ubuntu Package Importer" is a robot). We browsed the profile page of each member and recorded the number of people affiliated with Debian.

[4]We used "open source software" and "leadership" as keywords in Google Scholar to search for papers published in primary journals in information systems, management, and economics from 1998 to 2018. Table A1 in the appendix lists the seminal papers discussing the leadership/characteristics of leaders in an OSS project/community.

used an onion model to depict different roles (i.e., initiators, core developers, codevelopers, active users, release coordinators, and passive users) and their interrelationships. Aberdour (2007) simplified this model into four roles, namely, core team, contributing developers, bug reporters, and users. Setia et al. (2012) viewed OSS participants on the basis of two main roles: core developers who contribute to designing and developing the OSS, and peripheral developers who work to enhance and popularize the focal OSS. Continuing from the role difference, the second commonality is the role-based distance to the OSS project leader. In the onion model, such distance is visually observable, with the project leaders in the inner layer, the developers in the middle layer, and the other participants in the outer layer (Aberdour 2007; Crowston and Howison 2006).

On the basis of such simplification in the roles of OSS participants, we define two types of followers of OSS project leaders: developers and observers (Aberdour 2007; Chengalur-Smith et al. 2010; Crowston and Howison 2006; Setia et al. 2012). They can be distinguished by two facets: their different contributions to OSS projects and their different distance from leaders. The former facet has been articulated previously (Aberdour 2007; Crowston and Howison 2006). Developers who are directly engaged in creating software (i.e., the inner layer of the onion model) are essential contributors to an OSS project (Mockus et al. 2002), while observers, who are the indirect contributors, maintain an active interest in the development progress and provide comments and feedback to refine or perfect the OSS (Chesbrough 2003; Dahlander and Magnusson 2005). The latter facet suggests that developers and observers also differ in their distance from the OSS project leader (Aberdour 2007; Crowston and Howison 2006). On the basis of previous literature that conceptualizes leader–follower distance as a dyadic relationship (Napier and Ferris 1993), Antonakis and Atwater (2002) argued that the distance hinges on the extent to which information (from the leader) is accessible to followers.

Despite the role-based difference between developer and observer, the motivations in followership of the two roles converge. Instead of seeking immediate monetary return or reward, that is, reward-based motives, OSS followers are interested in their own self-development (Blincoe et al. 2016; Dvir et al. 2002; Li et al. 2012). Learning orientation is a primary motivation for OSS participants to subordinate themselves to a leader, through which the followers are empowered to develop their competence (Gong et al. 2009; Hannah and Lester 2009; Ilies et al. 2005). Thus, two types of learning approaches, namely, enactive mastery experience (EME) and mastery modeling (MM), can prevail in the leader–follower relationship (Bandura 1977; Gong et al. 2009).[5] The former, which is typically obtained by playing the role of a developer, refers to the direct experience of attaining a task or practice during contact with the leaders; the latter refers to an observational learning experience from the leader as a proficient model, which can be obtained by serving as an observer.

Followership Enables Participants to Learn, and Learning Promotes Code Reuse

Table 1 compares developer and observer followers. We argue that an OSS leader with diverse developers can foster (her or his dedicated) code to be reused in other projects through the involvement of these developers, whereas an OSS leader with many observers can foster code reuse through such observers' interactions with and influence on other projects.

Developers as Followers

In developing OSS, a project leader may work with more developers who have previous collaboration experience and work less with developers who have no previous collaborative experience or vice versa (Antonakis and Atwater 2002; Walker et al. 1994). Thus, an OSS project can be visualized through a continuum of followership, with one end indicating close developer collaboration and the other indicating open developer collaboration. An OSS project characterized by close developer collaboration has a high proportion of developers who have previous collaborative relationships with the focal project leaders. Conversely, a project marked by open developer collaboration reflects exoterically unspecified collaborations between the focal project leader(s) and the developers (Baldwin and von Hippel 2011; Belenzon and Schankerman 2015). Both collaboration mechanisms have their proponents, which are elaborated subsequently.

According to followership theory, close developer collaboration can be conducive given individuals' tendency to work well with those they have previously collaborated with (Hinds et al. 2000). Previous OSS studies have supported this notion. For example, Hahn et al. (2008) observed a developer's decision to participate in an OSS project is correlated with his or her personal experiences with coworkers in that project, such as whether he or she knows these coworkers or has had previous collaborative experiences with them. Several other

[5]In our context, *enactive mastery experience* refers to learning through direct engagement in the cocreation of OSS projects, while *mastery modeling* denotes active observational learning from prolific models, such as the leaders.

Table 1. Comparison of Developer and Observer Followers

	Developer as Followers		Observers as Followers
Learning Approach	Enactive mastery experience: In proximately working with leaders, developers progressively cultivate strong commitment with leaders and proactively dedicate themselves to developing their competence (Gong et al. 2009; Hannah and Lester 2009; Ilies et al. 2005).		Mastery modeling: Observers effectuate general learning purposes by observing and accumulatively gaining miscellaneous knowledge through attaching OSS project leaders (Blincoe et al. 2016).
Key Mechanism Related to the Learning Process	**Proponents for Close Developer Collaboration**	**Proponents for Open Developer Collaboration**	**Impact of Observer**
Quality of Knowledge/ Information	Repetitive collaboration among developers results in improved information quality, which is important in the future performance of the OSS project (Grewal et al. 2006).	Open collaboration facilitates knowledge spillover across OSS projects, which leads to out-performance (Fershtman and Gandal 2011; Singh et al. 2011). Open structure facilitates knowledge appreciation, which can benefit the OSS project (Breschi and Lissoni 2009; Horta et al. 2010).	An increased number of observers can facilitate the proliferation of OSS project knowledge base (Franke and von Hippel 2003; Lakhani and von Hippel 2003).
Locus of Knowledge/ Information	Repetitive collaboration enables OSS developers to have great access to public knowledge due to relatively low barriers in communication and coordination (Hahn et al. 2008).	Open collaboration makes OSS developers have dissimilar strategies for knowledge seeking, which can effectively avoid convergent knowledge. Consequently, the discovery of new knowledge artifact is conducive to proliferating knowledge sharing and code reuse in the community (Desouza et al. 2006). The overall diversity in the OSS project team, which is achieved through open collaboration, can facilitate knowledge exchange (Daniel et al. 2013).	Popularizing the OSS projects in the community relies on the presence of observers. Thus, having a large number of observers is important to achieve this goal (Setia et al. 2012).

works have advocated the notion that OSS developers prefer to work with those they have formerly collaborated with, which reduces uncertainty and increases access to information and other crucial resources for OSS development (e.g., Grewal et al. 2006; Hinds et al. 2000).

Although OSS developers have high inclinations to work with OSS project leaders across projects, such repetitive collaboration seemingly contradicts the antecedents for the creation of innovation in previous literature (Daniel et al. 2013; Skilton and Dooley 2010). Breaking collaborative boundaries and injecting outsiders into the team are instrumental to mitigate this drawback. Thus, an open collaboration structure is recommended because it promotes the entry of new developers who can eventually disseminate the code or knowledge into different projects, thereby ultimately promoting code reuse.

Theoretically, knowledge can be transferred from one project to another through either direct or indirect spillover (Desouza et al. 2006; Fershtman and Gandal 2011; Singh et al. 2011). For the former, knowledge is transferred from one project to another through shared developers. Developers may directly reuse the knowledge (i.e., source code in an OSS context) that they learn from one project and apply it to another. This can explain why close to 45% of the Ubuntu core development team members are also affiliated with the Debian project. For the latter, a developer working on one OSS project may take his or her knowledge to a second project with which he or she is affiliated and share it with participants in the second project, who may in turn reuse it when working on a third project.

The positive bearing of open collaboration on code reuse can also be understood by the point that OSS followers subordinate themselves to a leader. This leader empowers them to develop their competence and acquire new knowledge by working with the leaders (Gong et al. 2009; Hannah and Lester 2009; Ilies et al. 2005). Previous followership literature has argued that when followers are engaged in a leader–follower relationship with the purpose of learning, they cultivate a strong individual commitment to what they have dedicated to learning (Dvir et al. 2002). Hence, these developers apply what they have learned to other projects, which can take the form of code reuse.

Thus, the developers who autonomously follow leaders for self-development ensure their learned knowledge is appreciated. Knowledge appreciation can be achieved through gratuitous knowledge exchange spanning interorganizational boundaries (Breschi and Lissoni 2009; Horta et al. 2010). Conversely, maintaining a collaborative relationship (i.e., closed collaboration) shrinks the circle of the knowledge's application (i.e., occurrences of code reuse in other projects), which eventually leads to knowledge depreciation and less recognition of the individual developers (Horta et al. 2010). Both consequences are detrimental to the extent of code reuse of the focal OSS project. Therefore, we posit

> **Research Conjecture 1**: *OSS leaders who engage in fewer repetitive (i.e., open) collaborations with the same people across multiple OSS projects will experience greater code reuse.*

Observers as Followers

Compared with developers, observers are distant followers. They can enhance or popularize an OSS project by spreading knowledge or information about it through social interactions with others in the community (Franke and von Hippel 2003; Lakhani and von Hippel 2003; Setia et al. 2012). The assumption about developers described in Research Conjecture 1 may not be applicable for observers because they have different ways of contributing and a dissimilar leader–follower distance. Given a close relationship with the OSS project leader and the nature of the contribution (i.e., writing code), a developer's commitment or responsibility toward a certain leader can be manifested in terms of reusing code in OSS projects to which she or he is concurrently dedicated. Such an initiative for promoting code reuse may not work well for the observers.

Observers adhere to the OSS project leader in a different manner from developers. An observer who "follows" an OSS project leader tracks news and updates on a person handling multiple projects in the OSS community. Blincoe et al. (2016) surveyed observers in GitHub to understand their motivations in becoming observer followers of a person and outlined seven key motives. Excluding 18.8% of the responses, which indicated "no benefits" from becoming an observer in GitHub, nearly all of the respondents expressed their following motivation to be related to learning. Their motivation spans from general learning purposes such as receiving updates and information from the OSS projects worked on by the leaders (64%), to focusing on code for gaining technical knowledge and collaborative opportunities (21.72%), and to miscellaneous interests in gaining access to OSS information (14.28%).

When the observers are affiliated with other OSS projects, they may share the acquired knowledge from projects they follow with different projects in various forms, such as by suggesting code solutions from their followed projects to problems faced by other projects (von Krogh et al. 2005). Apart from such direct application of their learned knowledge, observers can also share what they have learned with people in other projects, who will in turn reuse such knowledge in the third OSS project in terms of code reuse. Stewart (2005) found that the opinions of a member with high status in the OSS community (i.e., an OSS project leader) disproportionally affect relatively low-status members, such as their observer followers. Thus, observers comply with the will of their leaders and perform desirable behaviors as well. For these reasons, observers can serve as social resources to ensure that OSS projects proliferate (Gulati 1999; Lavie 2006).

As depicted previously, developers proximately work with OSS project leaders and learn through EME. Applying their learned knowledge to other projects promotes knowledge diffusion, which is manifested by code reuse. For observers, they assimilate miscellaneous information from their followership. Popularizing and spreading such knowledge

throughout the community facilitate code reuse. Thus, we argue that increasing the number of observer followers (of an OSS leader) can expand the pool of people with knowledge about the followed project, which eventually increases the likelihood that learned knowledge will be shared and that code will be reused in other projects.[6] Thus, we posit

> **Research Conjecture 2**: *OSS leaders who have a larger number of observers will experience greater code reuse.*

Method

The proposed conjectures were validated using a set of data collected from GitHub, an online community-based platform for OSS projects. We selected GitHub over traditional OSS forges such as Sourceforge.net (Sen et al. 2012; Singh et al. 2011) and over specific, well-publicized OSS projects such as Debian (O'Mahony and Ferraro 2007), Apache (Lakhani and von Hippel 2003), and Mozilla (Mockus et al. 2002) because of GitHub's capacity to develop its platform and because of its goal to provide a societal environment conducive to software innovation. In terms of analyzing the personnel resources affiliated with each hosted OSS project, GitHub records a list of activities of the registered members. Furthermore, GitHub has a microblogging feature that offers a neutral social environment through which observers can investigate each individual or project. Specifically, the "follow" function in microblogging provides an unbiased societal environment for observers (excluding coworkers). With regard to monitoring of code reuse, GitHub has a novel revision-control system that tracks all codes that are reused in other projects (Loeliger and McCullough 2012). Through this system, OSS developers can adopt codes from other projects to build their own projects (Bird et al. 2009). GitHub measures the frequency of code reuse rather than the number of downloads. This feature directly and effectively reflects the success of open innovation (Haefliger et al. 2008).

Sample and Data Collection

The dataset used for the analysis was collected through the application programming interface provided with explicit permission from GitHub. The data were collected twice, in the winter of 2012 (T_1) and in the winter of 2014 (T_2). The two-year lag in the data collection enabled the validation of the conjectures without the endogeneity issues typical of cross-sectional data. In addition, longitudinal observation helped identify whether the OSS projects were ongoing or inactive (i.e., hosted on the platform for years without any changes), which was useful given the context in which numerous ungoverned projects such as student homework and test files were floating in the OSS community (Rainer and Gale 2005). In the initial stage, T_1, 1,043 OSS projects were randomly selected. If the leader of an OSS project did not concurrently manage more than two such projects, then that project was removed from the dataset for T_1 because the developer's followership could not be determined. Consequently, 332 projects were removed.

Next, in terms of the two-year observation period, 61 of the remaining 711 OSS projects were excluded from further analysis for three reasons. First, 32 of the projects were removed because they were inactive, that is, they had not been updated within the two-year observation period. Next, 12 projects were independently removed by their leader(s). Finally, after the projects' pages were collected (T_1) and carefully reviewed, 17 projects were removed because their leaders had declared "Stop updating" on the projects' pages, although those projects were neither removed nor closed.

The final dataset consisted of 910 usable observations from 650 projects codesigned by 179 leaders and 5,659 developers. Each OSS project could be managed by a single leader or co-administered by multiple leaders.[7] The leader(s) of each OSS project were identified, and these data became the antecedent for the construction of the proposed followership of the developers and observers. In addition, this identification enabled a precise level of analysis to measure the dynamics of each leader's followership and validate the research conjectures.

Dependent Variable

Code reuse occurs when a copy of the source code is transferred from a software package to an independent project (Robles and González-Barahona 2012). For each OSS project j administered by a leader i, the extent of code reuse is denoted by $CodeReuse_{ij}$, and the computed difference between the values at T_1 and T_2 is denoted by $Diff\_CodeReuse_{ij}$. Given that each project leader had to administer at least two projects in the research setting, the average value of $Diff\_CodeReuse_{ij}$ was computed for each leader as a dependent variable:

[6]To avoid confusion, we should briefly explain why the two conjectures are phrased differently. Specifically, whereas developer followership is formed from project-related activities (i.e., project-based cooperation with "leaders"), observer followership is not (i.e., it is shaped simply by following others to gain knowledge or information). Because of this difference, Research Conjecture 1 focuses on proportions of followers spanning OSS projects, whereas Research Conjecture 2 does not. We return to this issue later.

[7]The unit of analysis was the leader of the project. Thus, one project with multiple leaders was expanded into multiple observations in the data sample.

$AvgDiff\_CodeReuse_i$. The resulting longitudinal measurement can best reconcile the endogeneity problem attributed to the causal relationship between followership and reuse frequency.

Independent Variables

Developers as Followers

The degree of closeness of the developer–collaboration form is reflected in the dyadic structural intensity between an OSS project's leaders and developers; the degree of closeness is formulated as

$$Developer\_followership_{ijt} = \frac{\Sigma Developers_{(sin|j \notin n)}}{\Sigma Developers_{sj}}$$

where j denotes an OSS project, i is a leader of the OSS project j, s is a developer, t is a timestamp, and n is the set of OSS projects managed by leader i.

The preceding equation indicates the dyad between the leaders of the focal project and their collaborating developers. Specifically, the value denotes the proportion of developers who contribute to other OSS projects with leaders who are the same as those of project j. A high value for $Developer\_followership_{ijt1}$ denotes that leader i has a relatively stable set of developers across OSS projects, indicating a relatively tight collaborative followership. The data from the Git[8] commit record for each OSS project were used to construct the developer–collaboration measurement. In OSS development, an update to the OSS project source code denotes a commit, for which detailed information about the submitter, such as identity and time, is recorded. By using the data collected at T_1, the complete commit information is obtained for each OSS project in the dataset, from its inception stage to T_1. This information was used to determine whether the focal developer had worked for multiple projects administered by the same people. By using the formula described, the developer–collaboration form for each leader was computed; it is denoted as $Developer\_followership_{ijt1}$.

Observers as Followers

The measurement of observer followership was directly obtained from the number of observers for each leader. The dataset was coded at the leader level rather than at the project level. For example, in an OSS project governed by two leaders, the number of observers following each leader is regarded as an observation in the dataset ($Observer\_followership_{it1}$).

Control Variables

Age-Related Covariates

The age-related risk of an institution's mortality has been extensively examined in the organizational ecology literature. Several contesting explanations regarding age-related factors exist, including the liability of newness (which states that a project has a high risk of failure at the initial stage but that this risk eventually declines) and the liability of aging (in which the risk of failure increases as organizational age progresses) (Chengalur-Smith et al. 2010; Grewal et al. 2006; Stewart et al. 2006). An OSS project, which is an online virtual organization, faces the same risks. Thus, age-related covariates should be considered. Followership can fluctuate over time (Baker 2007); thus, the ages of OSS project j and its leader i at T_1 were included as control variables $Project\_age_{jit1}$ and $Leader\_age_{ijt1}$, respectively.

Leader-Related Covariate

The observer followership can also be inversely established. The leader of an OSS project can proactively follow other users on GitHub, in return for which certain observer resources can be reciprocally augmented (Riquelme and Gonzalez-Cantergiani 2016). In the analysis model, we included the number of people whom the leader i of the OSS project j had followed at T_1 ($Leader\_following_{ijt1}$, a control variable) to account for an inversely obtained observer followership.

Project Demographics

The project demographics are composed of two covariates: the size of the OSS project j (managed by leader i) at T_1 ($Project\_size_{jit1}$) and the availability of that project's knowledge kit at T_1 ($Project\_knowledge_{jit1}$). The project size (measured in kilobytes) was expected to explain the complexity associated with the software as the innovation developed (Singh et al. 2011). The knowledge kit is where a project stores and shares knowledge about its product. This kit contains descriptive information related to the project, such as an introduction or the project's goal, rather than its development logs. Thus, the knowledge kit is primarily used for providing basic information (for users) or internal documentation and

[8]Git is a version control system used in GitHub.

communication (for within-project developers and development teams) (Hahn et al. 2008). This also influences the documentation quality (i.e., an important dimension in measuring the quality of an OSS project), as suggested in a previous study (Crowston et al. 2006). Wikis are often used to provide basic information for users to download. A binary variable was used in the current study to code the availability of the knowledge kit.

Development-Related Covariates

Three project-level covariates were utilized to indicate the magnitude of development. The first was the cumulative number of versions for OSS project *j* (managed by leader *i*) at T_1 ($Cumulative\_version_{jit1}$). A high value for $Cumulative\_version_{jit1}$ indicates that an OSS project has been widely updated; this measure indirectly reflects the extent of the contributions to this project (Singh et al. 2011). $Cumulative\_version_{jit1}$ can reflect the amount of work done on a project; it is linked to the project's technical quality (Singh 2010). The second covariate was the developer-to-leader ratio of OSS project *j* (managed by leader *i*) at T_1 ($Ratio_{jit1}$). This variable was created to control the approximate general workload of the developers on each project. Most OSS projects are intentionally operated using meritocracy, in which promotions, such as that from developer to leader, hinge on the extent of one's contribution. However, O'Mahony and Ferraro (2007) found that such contributions comprise technical and organization-building contributions. Compared with people on the leadership team, developers may undertake more technical contributions, particularly in coding. Thus, a high value can refer to great technical capability that the project fosters. Thus, active developers were regarded in this study as frontline workers who directly influenced the final creation. Consequently, the developer-to-leader ratio needed to be controlled. The third covariate was the recency of OSS project *j* (managed by leader *i*) at T_1: $Recency_{jit1}$, which is calculated as the difference (in days) between T_1 and the publish date of the most recent version. This value is used to control the recency of the OSS's latest changes (Braunschweig et al. 2012).

Observer-Related Covariate

As in the followership relationship among individuals, people can show interest in following the news and updates about an OSS project. Such observers were regarded as supplementary resources for these projects (Dabbish et al. 2012). Thus, the control variable $Project\_observer_{jit1}$, which denotes the number of observers for OSS project *j* (managed by leader *i*) at T_1, was used as a covariate to control potential heterogeneity.

Technology-Related Covariates

The previous literature has demonstrated that programming languages and project types are important considerations in an OSS project (Zhu and Zhou 2012). For example, there are likely more developers who know popular programming languages, such as Java or PHP, than less popular languages. The dataset used for this research comprised 26 programming languages[9]; the categorical variable $Programming\_language_{ji}$ was used to denote the programming language for OSS project *j* (managed by leader *i*). Apart from the programming languages, the project types of OSS also had to be controlled given the potential restrictions on code reuse to similar or identical categories. Unlike Sourceforge.net, which clearly categorizes the types of each OSS project, GitHub has no precise label for each OSS project. For complying with standard coding procedures, two independent research assistants with computer science backgrounds were recruited to label the OSS projects in the dataset manually by reading the documentation for each OSS project (Palvia et al. 2007). Eventually, the following three categories were identified: Software Framework,[10] Application Software,[11] and Documentation File.[12] Thereafter, two coders were recruited to assign the OSS projects into the three categories. The Cohen's Kappa coefficient for these classifications was 0.87, indicating substantial inter-coder reliability. Accordingly, a categorical variable, $Project\_category_{ji}$, was used to indicate the category of OSS project *j* (managed by leader *i*). We also referred to the TIOBE index[13] (Paulson 2007) to control the popularity of each language in OSS project *j* (managed by leader *i*) at T_1 ($Language\_popularity_{jit1}$) because people are attracted to OSS projects written in popular programming languages. To further control the influence of language popularity on code reuse, we used Factiva,[14] a global news database, to investigate the media coverage of each programming language. The popularity of a programming language

[9]The 26 programming languages are Ruby, JavaScript, Python, Objective-C, Nu, Perl, C, Go, CoffeeScript, Erlang, Scheme, Emacs Lisp, Io, PHP, Shell, VimL, C++, Clojure, Java, Haskell, Scala, Objective-J, ActionScript, Lua, Common Lisp, and Arc.

[10]The category of "Software Framework" is a software abstraction providing generic functionality that users can individually change through additional code providing particular functionality.

[11]The category of "Application Software" is a set of one or more programs designed to perform the operations for a specific application.

[12]The category of "Documentation File" is a simple static text file that is related to software development, such as a Cascading Style Sheet or configuration file.

[13]https://www.tiobe.com/tiobe-index/

[14]https://www.dowjones.com/products/factiva/

in the media may alter individuals' decisions to become involved with OSS projects that use that language (Bennett 2003). Thus, we counted the number of pieces of news that mentioned the language in the three months prior to T_1 ($Language\_media\_coverage_{jit1}$).

Data Analysis

Main Results

Tables 2 and 3 present the descriptive statistics and the correlation matrix of the variables, respectively. Table 3 shows that all correlation coefficients are less than 0.6, which indicates a lack of multicollinearity. The variance inflation factors (VIFs) of the predicting variables were also computed, and indicated no multicollinearity; they are described in the appendix.

Multilevel regression was used to validate the two proposed conjectures. Table 4 presents the complete results. The unit of analysis was the individual leader for each OSS project. Thus, the multilevel regression was modeled with the OSS project nested within the OSS leader. The number of observers for an individual leader was denoted as $Observer\_followership_{it1}$. For the developers acting as followers ($Developer\_followership_{ijt1}$), an individual may lead multiple OSS projects, and each such project may entail a distinct developer collaboration (ranging from open to closed). Accordingly, in the data analysis, $Observer\_followership_{it1}$ was specified as a fixed effect, and $Developer\_followership_{ijt1}$ was specified as a random effect. The categorical variables $Programming\_language_{ji}$ and $Project\_category_{ji}$ were created and included as dummy variables in the regression model but are not reported in Table 4 for brevity in accordance with practice (Singh et al. 2011).

Model 1 in Table 4 is the baseline model, which includes only the intercept term. Model 2 adds the control variables. The independent variables, $Developer\_followership_{ijt1}$ and $Observer\_followership_{it1}$, were added to the regression in Models 3 and 4, respectively. Model 4 is the full model, which includes $Developer\_followership_{ijt1}$ and $Observer\_followership_{it1}$. The deviance is the negative of twice the log-likelihood; it has a chi-square distribution with the difference caused by increasing the number of coefficients across models. The reduction in deviance is explained by the deviance values for Models 2, 3, and 4, which are all significant ($p < 0.01$). The coefficients were estimated with random slopes by using maximum likelihood.

In the case of developer followers, the significant negative coefficient of $Developer\_followership_{ijt1}$ indicates that a relatively open developer collaboration can facilitate code reuse in a focal project leader's other OSS projects. A project with a leader of a relatively stable set of developers across OSS projects has a low extent of code reuse in other projects (as stated in followership theory). Thus, ***Research Conjecture 1 is supported***. Interestingly, the developer-to-leader ratio positively influences the extent of code reuse. This supports previous findings in which the developer was found to be the most valuable resource in an OSS project (Shah 2006; von Hippel and von Krogh 2003). In the case of observer followers, the results suggest that the number of observers following an individual influences the extent of code reuse in that person's OSS projects, ***which supports Research Conjecture 2***. We interpret the result as implying that having more observer followers (of an OSS leader) can lead to more people learning about the followed project, which eventually increases the likelihood that learned knowledge will be shared and that code will be reused in other projects. We also checked if the significant results were elicited from the control covariates by running the models after excluding the control variables. The results of first conjecture (coefficient = −0.162, $p \leq 0.05$) and second conjecture (coefficient = 0.201, $p \leq 0.01$) remained consistent.

Prior to further interpreting the results and presenting the robustness checks, we briefly restate the reasons for using different measures for the two types of followership. As argued previously, developer and observer followers have different ways of learning and applying what they learn to other projects, as manifested by code reuse. Whereas developer followership stems from collaborative work with a leader in certain project(s), observer followership is independent of OSS projects (i.e., it can be formed without engaging in OSS development.[15] As a result, we cannot measure observer followership in the same way as developer followership because the former's followership does not span OSS projects.[16]

[15]Observer followership can be established by following a particular user (OSS leader in our context) in Github (Blincoe et al. 2016). Thus, whether this observer engages in OSS projects governed by the leader does not matter.

[16]Even though a lack of symmetry between the two measures of followership is appropriate in this paper, we ran a test to check if the results would vary if more symmetrical measures were used. To do so, we used the same measure of observer followership to construct the measure of developer followership, denoted by ($\boldsymbol{Developer\_followership2_{ijt1}}$), by counting the number of developer followers spanning OSS projects. This number could be small if a leader possessed strong followership. The following scenario was assumed: one leader manages two different projects with the same developers (three developers per OSS project), and another leader manages two different projects with different developers (also three developers per OSS projects). Under this measure of developer followership, the former has three developer followers, but the latter has six developer followers. A small number indicates a strong followership. Compared with our measurement, that is, the proportion of people who collaborate, this measure has less validity because a small number can also represent a small-sized project. Although the symmetrical measure accentuates uniformity, the validity of developer followership is also challenged. Nevertheless, by applying the same regression model, we obtained consistent results (coefficient = −0.114, $p \leq 0.01$), which should provide even greater confidence in our results.

Table 2. Descriptive Statistics

Variable	Variable Description	Mean	Std. Dev.	Min	Max
$AvgDiff\_CodeReuse_i$*	Average difference in the extent of code reuse between T_2 and T_1 for OSS projects managed by leader *i*	1.088	1.067	0	6.741
$Developer\_followership_{ijt1}$	Extent of collaborative followership of OSS project *j* managed by leader *i* at T_1	0.320	0.371	0	1
$Observer\_followership_{it1}$*	Extent of leader *i*'s followership among observers at T_1	5.789	1.880	0	9.031
$Project\_age_{jit1}$	Age of OSS project *j* (managed by leader *i*) at T_1	1058.683	522.600	6	1784
$Leader\_age_{ijt1}$*	Age of leader *i* (managing OSS project *j*) at T_1	7.463	0.122	5.342	7.533
$Leader\_following_{ijt1}$*	Number of people followed by leader *i* of OSS project *j* at T_1	2.901	1.652	0	5.826
$Project\_size_{jit1}$*	Size of OSS project *j* (managed by leader *i*) at T_1	5.769	1.677	0	13.050
$Project\_knowledge_{jit1}$	Availability of the knowledge kit for OSS project *j* (managed by leader *i*) at T_1	0.871	0.335	0	1
$Cumulative\_version_{jit1}$*	Cumulative versions of OSS project *j* (managed by leader *i*) at T_1	3.449	1.926	0	10.226
$Ratio_{jit1}$	Developer-to-leader ratio of OSS project *j* (managed by leader *i*) at T_1	27.673	104.474	0	1309.5
$Recency_{jit1}$	Difference (in days) between T_1 and the publishing date of the most recent version of OSS project *j* (managed by leader *i*)	111.138	135.757	4	423
$Project\_observer_{jit1}$*	Number of observers of OSS project *j* (managed by leader *i*) at T_1	2.453	1.923	0	8.545
$Programming\_language_{ji}$**	Types of programming languages used in OSS project *j* (managed by leader *i*) at T_1	18.774	5.334	1	26
$Project\_category_{ji}$**	Category of OSS project *j* (managed by leader *i*) at T_1	1.813	0.836	1	3
$Language\_popularity_{jit1}$*	TIOBE index of programming language popularity for OSS project *j* (managed by leader *i*) at T_1	0.022	0.031	0.001	0.179
$Language\_media\_coverage_{jit1}$*	Media coverage of the programming languages used in OSS project *j* (managed by leader *i*) at T_1	6.280	2.487	0	8.769

*Logarithmic transformed value. **Categorical variables.

Table 3. Correlation among Variables

		1	2	3	4	5	6	7	8	9	10	11	12	13
1	$Developer\_followership_{jt1}$	1												
2	$Observer\_followership_{it1}$	0.211	1											
3	$Project\_age_{jit1}$	0.200	−0.061	1										
4	$Leader\_age_{ijt1}$	0.198	0.402	0.060	1									
5	$Leader\_following_{ijt1}$	0.011	0.345	0.102	0.223	1								
6	$Project\_size_{jit1}$	−0.119	−0.073	0.017	−0.033	−0.050	1							
7	$Project\_knowledge_{jit1}$	0.134	−0.166	0.039	−0.015	−0.184	0.005	1						
8	$Cumulative\_version_{jit1}$	−0.354	−0.031	−0.094	−0.023	−0.044	0.355	−0.196	1					
9	$Ratio_{jit1}$	−0.145	0.094	0.113	−0.098	0.092	0.033	−0.239	0.181	1				
10	$Recency_{jit1}$	0.020	−0.150	−0.114	0.140	0.136	−0.042	0.057	−0.048	−0.201	1			
11	$Project\_observer_{jit1}$	−0.152	0.007	0.461	−0.184	0.045	0.082	−0.174	0.198	0.573	−0.536	1		
12	$Language\_popularity_{jit1}$	0.031	−0.002	0.059	−0.001	0.006	0.018	−0.004	−0.028	0.054	−0.023	0.064	1	
13	$Language\_media\_coverage_{jit1}$	−0.005	−0.077	−0.059	−0.050	−0.037	−0.059	0.021	−0.032	−0.002	0.046	−0.034	0.242	1

Table 4. Main Results (910 Observations)

Dependent Variable	$AvgDiff\_CodeReuse_i$			
	Model 1	Model 2	Model 3	Model 4
$Developer\_followership_{ijt1}$	—	—	—	−0.197** (0.083)
$Observer\_followership_{it1}$	—	—	0.215*** (0.017)	0.221*** (0.017)
$Project\_age_{jit1}$	—	−0.084 (0.070)	0.031 (0.065)	0.075 (0.068)
$Leader\_age_{ijt1}$	—	−0.400 (0.249)	−1.568*** (0.248)	−1.518*** (0.248)
$Leader\_following_{ijt1}$	—	0.005 (0.018)	−0.057*** (0.018)	−0.062*** (0.018)
$Project\_size_{jit1}$	—	−0.019 (0.020)	−0.012 (0.019)	−0.012 (0.019)
$Project\_knowledge_{jit1}$	—	−0.357*** (0.095)	−0.252*** (0.088)	−0.243*** (0.088)
$Cumulative\_Version_{jit1}$	—	−0.014 (0.020)	−0.011 (0.018)	−0.024 (0.019)
$Ratio_{jit1}$	—	0.113*** (0.043)	0.010** (0.040)	0.095** (0.039)
$Recency_{jit1}$	—	−0.110*** (0.025)	−0.030 (0.024)	−0.035 (0.024)
$Project\_observer_{jit1}$	—	0.154*** (0.028)	0.152*** (0.026)	0.145*** (0.026)
$Language\_Popularity_{jit1}$	—	0.448 (0.958)	0.250 (0.890)	0.313 (0.888)
$Language\_Media\_Coverage_{jit1}$	—	−0.004 (0.012)	0.004 (0.011)	0.005 (0.011)
Intercept	1.043*** (0.038)	4.525** (2.083)	11.789*** (2.009)	11.403*** (2.010)
Deviance (−2 × log-likelihood)	2748.763	2439.049	2293.610	2288.057
Deviance difference	—	309.71	145.438	5.553
Akaike's information criterion	2756.763	2521.049	2377.611	2374.057
Bayesian information criterion	2776.230	2720.593	2582.022	2583.335

*p ≤ 0.1; **p ≤ 0.05; ***p ≤ 0.01; figures are displayed in the format of "Coefficient (Std. Err.)"

Note: We applied the stepwise approach for estimating the coefficients. Model 1 is the baseline model, in which only the intercept is included. Model 2 contains all control covariates. Two independent variables, $\boldsymbol{observer\_followership_{it1}}$ and $\boldsymbol{Developer\_followership_{ijt1}}$, are entered into Models 3 and 4, respectively. In summary, Model 4 is the full model. To easily interpret the results, we scaled down $Project\_age_{jit1}$, $Ratio_{jit1}$, and $Recency_{jit1}$ by factors of 1000, 100, and 100, respectively. The programming languages, denoted by $Programming\_Language_{ji}$, and project category, denoted by $Project\_Category_{ji}$, were created as dummy variables and estimated in the regression model but are not reported for brevity.

In addition to the main results, we obtained other interesting findings, such as for the presence of the knowledge kit. Specifically, $Project\_knowledge_{jit1}$ has a detrimental effect on code reuse. A plausible explanation is that an OSS project's knowledge kit only contains brief descriptive information, such as an introduction and the project's goals, and does not include information such as development logs that will facilitate code reuse. A knowledge kit is primarily used for providing basic information about a project (for users) or internal documentation and communication (for within-project developers and development teams). Knowledge kits are not helpful to individuals who are researching how to reuse specific code segments. Thus, the provision of the knowledge kit may be detrimental to code reuse, and those who hope to acquire code-reuse clues by accessing the available wikis can be disappointed. The two variables controlling the popularity of the programming languages are insignificant, which rules out the potential threat that language popularity will affect code reuse.

Post Hoc Analyses and Additional Investigations

We conducted *post hoc* analyses to verify the main results and rule out alternative explanations. We also conducted additional investigation, including a survey and a series of focus groups, to triangulate the findings, provide evidence regarding our conjecture of the presence of learning, and present further insights.

We conducted four sets of *post hoc* analyses to enhance the robustness of preceding findings and rule out the potentially alternative explanations (see the appendix for the details). The first set was based on two robustness checks of the results reported in the previous section. Specifically, we varied the operationalization of the dependent variable and the regression analysis models to check the empirical consistency. The second set of *post hoc* analyses was conducted by segmenting the OSS projects into affiliated categories that could have diverse complexity in OSS development (Au et al. 2009; Sen 2007). We attempted to unveil why two types of followership had different impacts on the code reuse across different natures of OSS projects. The third set of *post hoc* analyses used ordinary least squares analysis with stepwise approach to check for potential multicollinearity problems. The fourth set of *post hoc* analyses was conducted to rule out an alternative explanation for "follower inbreeding" among certain OSS projects.

Apart from the *post hoc* analyses, we also conducted a survey and a series of focus groups to confirm the presence of learning among followers (see the appendix for the details). Specifically, our conjectures were based on the idea that followership is important for code reuse because it provides a way for participants to learn. Moreover, we described two ways of learning: EME and MM. We therefore conducted the survey to test our suggestion that *learning from OSS followers* can explain the relationship between followership and code reuse. As shown in the appendix, the results of our survey of 164 registered GitHub members support this idea: EME (estimated coefficient = 0.24, p value $\leq$ 0.01) and MM (estimated coefficient = 0.15, p value $\leq$ 0.05) are both positively associated with the intention of code reuse. Our qualitative findings from focus group interviews enrich the survey findings. For example, we identify that EME can be consolidated from (1) the interaction between developer followers and OSS project leaders and (2) the collaborative diversity. We also affirm that the observer followers learn from following leaders and spontaneously reuse their acquired information or knowledge in their own projects. In addition to confirming the presence of learning, additional insights such as motives of code reuse and sharing mentality among the OSS participants were discussed in the focus group interviews. These qualitative findings offer numerous avenues for future research.

Discussion

Overall, both of our conjectures were supported. We found that projects with a high variety of developers collaborating with the same leader(s) beyond the focal project experience high code reuse by other projects, as per Research Conjecture 1. This supports the idea in prior literature that compared to OSS projects that exhibit close developer collaboration, OSS projects that exhibit open developer collaboration benefit from the cross-fertilization that occurs with accumulating new ideas or knowledge from developers who may not have formerly worked with a leader (Kogut and Metiu 2001; Sawhney et al. 2005). Aside from such a conventional standpoint, our findings also provide complementary evidence explaining the positive role of open developer collaboration in code reuse from *learning* orientation.

Our results suggest that observers contribute to code reuse through a different scenario. Through such followership, observers can learn and accumulate various types of information or knowledge, such as general updates, trendy information, and coding or programming knowledge. Observers may then apply what they have learned to the other OSS projects through the manifestation of code reuse. In this way, projects with many observers who follow the leader of a focal project can promote code reuse for other projects, as per Research Conjecture 2.

Contributions to Theory

This study presents three contributions to OSS research and followership theory. First, it contributes to extant OSS research in two aspects: code reuse and application of followership theory in an OSS setting. Although the importance and prevalence of code reuse are recognized, to the best of our knowledge, little is known theoretically about promoting it (Haefliger et al. 2008; Sojer and Henkel 2010). Such a research gap is filled by this study in terms of the followership approach. We depict and characterize two types of follower roles of the OSS project leaders in promoting code reuse in the OSS context. Such a follower-centric approach to investigating the follower–leader relationship can be viewed as the second contribution to the OSS literature. Different from previous studies that mostly adopt a leader-centric approach, our study provides a different way of viewing relationships in online peer production. Specifically, our findings show the significance of active involvement from

followers in such relationships. This supports the argument in previous literature (Carsten et al. 2010; Haslam and Platow 2001) that followers should be regarded not only as passive recipients/subordinates but also as active followers (hence, active followership). In other words, we unveil that followers proactively perform promotion behaviors and cooperate with leaders to promote their desired outcomes.

Second, our study contributes by demonstrating the importance of understanding relational layers defined by role types. In OSS projects, the successive layers of roles depict the leader as the core, developers as the next layer, and observers as the outer layer (Aberdour 2007; Crowston and Howison 2006; Setia et al. 2012). Thus, our research findings concur with the general conceptual framework proposed by Uhl-Bien et al. (2014). This framework indicates that the consequential effects of followers on code reuse can vary depending on the roles played by followers. Our findings add to this body of knowledge by providing insights into the varying roles of followers depending on their distance from the leader. In the case of developers as followers, projects with a high composition of developers who have less prior collaboration with project leaders can be conducive to code reuse. Breaking the collaboration boundary and injecting outsiders into the team can be instrumental, although they may introduce a high cost of communication or coordination. This result adds to previous research on OSS that focuses on attracting and managing developers (Hahn et al. 2008; Li et al. 2012; Roberts et al. 2006; Shah 2006; von Krogh et al. 2012). In the case of observers as followers, having a large number of observers who are followers of the project leader(s) can facilitate code reuse. This study is innovative in that it reveals the role of observers in proliferating the OSS project. In online contexts, opinion leaders can spread news or information easily (e.g., via social media) to create influence via their numerous followers. As a result, OSS project leaders with substantial followers can advantageously export their innovation to many people.

Finally, this study examines the underlying mechanism related to followership and code reuse, thereby advancing our understanding of community-based governance. Specifically, we find that the learning orientation fostered by the followers is the key motive for followership and for promoting code reuse. OSS participants subordinate themselves to the particular OSS project leaders to learn and develop their own competence. Followers can then apply and share what they have learned from following project leaders, manifesting in code reuse. In summary, our findings reveal that learning objectives and self-development goals are key factors that ensure individuals collectively contribute to knowledge reuse and recombination. This adds to governance theory in online peer production.

Contributions to Practice

Practitioners can benefit from these findings in several ways. OSS leaders can benefit from learning how to expedite code reuse by other projects. Our findings suggest that project leaders might be able to do so if the staffing structure of their projects reflects an open collaboration structure. Furthermore, project leaders with a large number of observers are likely to benefit from code reuse by other projects. Accordingly, our findings can inform those OSS project leaders and help them recognize and leverage the value of their followers (e.g., to create and facilitate open collaboration and accumulate a large number of observers to promote code reuse).

Our findings also suggest that OSS followers, such as developers and observers, should reflect on their competence in learning from their followership and applying their learning to other OSS projects. As one interviewee (FG-2-F) from the focus group session stated, "*Reusing code from others is also a process of learning. In the course of reuse, I can absorb it well (the code and structure).*" For developers, avoiding repetitively working on OSS projects led by the same people is suggested, although such collaboration can save them a large amount of communication and coordination costs. As a respondent (FG-1-B) stated, "*Innovation and inspiration come from the diversity.*" Following an OSS project leader can help observers access information, which helps in accumulating information and knowledge.

For OSS platform owners, our findings highlight the importance of a flat hierarchy in the OSS community, in which knowledge reuse and recombination can be achieved (Dahlander and O'Mahony 2011). Thus, an appropriate organizational design to facilitate knowledge exchange in the community is encouraged for its implementation. In addition, our findings could help those redesigning website toolkits or "following" functions. For example, having a reputable or popular leader of an OSS project attracts additional observers, and having a leader who nurtures learning among followers is beneficial to inducing code reuse.

This study also offers insights for organizations that adopt an online user innovation community strategy (Di Gangi and Wasko 2009). Although this work examined an OSS community, the findings are also informative for other online user innovation communities, such as public innovation communities or internal ideation platforms. Companies can draw insights from the study to prevent their innovators from falling into silo mentalities. Our findings support characterizing followers by their learning orientations to facilitate knowledge reuse and recombination. Although these innovators are not contractually bound, they can eventually contribute to collective innovation.

Limitations and Suggestions for Future Research

This study has several caveats that provide opportunities for future research. First, although the GitHub platform offers a conducive open innovation platform for testing our research conjectures, the use of cross-platform data in future studies will allow researchers to compare the success of OSS projects. Although the key data sample (910 observations) was extracted from 650 OSS projects—many more than examined in the bulk of prior OSS studies—future research can consider the entire set of OSS projects in GitHub and other OSS platforms, including more comprehensive datasets (i.e., network variables or physical distance within each OSS project team or panel data) to validate these findings with efficient estimation. Second, as previously depicted, future studies can investigate code reuse from a dynamic approach instead of using a unidirectional method. According to the qualitative findings, the original contributors of code and the reusers can benefit from the process of code reuse. Next, this research focuses on the relationship between the followers of a project leader and code reuse. Although previous studies have supported the benefits of code reuse, the negative implications of code reuse are seldom discussed (Haefliger et al. 2008; Sojer and Henkel 2010). More research on this topic is needed. Finally, our additional investigations only examine the presence of learning in aggregate. Future research can distinguish different types of followers and characterize their learning approaches in code reuse. This would help further unveil the dynamics of learning in community-based production activities.

Conclusion

OSS, which is a form of open innovation in software development, has been discussed in multiple disciplines. Different from most prior studies that utilize a leader-centric approach to depict the follower–leader relationship and understand its impact on OSS project performance, this study adopts followership theory to investigate the significance of followers (who comprise a large proportion of the participants in an OSS project) in code reuse. Examining OSS projects through followership theory and OSS literature enabled this study to investigate two types of followers, namely, developers and observers, and their resultant influence on code reuse. The central finding is that followership is important for code reuse in OSS because it enables participants to learn, and learning promotes code reuse. This finding offers several important implications and numerous opportunities for future research.

Acknowledgments

The authors would like to thank the senior editor, Andrew Burton-Jones, the associate editor, and three anonymous reviewers for helpful comments provided throughout the review process of this article. The work described in this paper was partially supported by the National Natural Science Foundation of China (#71702133, #71704078, #71532015, #71801217) and City University of Hong Kong (CityU 11504417/9042571).

References

Aberdour, M. 2007. "Achieving Quality in Open-Source Software," *IEEE Software* (24:1), pp. 58-64.

Antonakis, J., and Atwater, L. 2002. "Leader Distance: A Review and a Proposed Theory," *The Leadership Quarterly* (13:6), pp. 673-704.

Au, Y. A., Carpenter, D., Chen, X., and Clark, J. G. 2009. "Virtual Organizational Learning in Open Source Software Development Projects," *Information & Management* (46:1), pp. 9-15.

Baker, S. D. 2007. "Followership: The Theoretical Foundation of a Contemporary Construct." *Journal of Leadership & Organizational Studies* (14:1), pp. 50-60.

Baldwin, C. Y., and Clark, K. B. 2006. "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?," *Management Science* (52:7), pp. 1116-1127.

Baldwin, C., and von Hippel, E. 2011. "Modeling a Paradigm Shift: from Producer Innovation to User and Open Collaborative Innovation," *Organization Science* (22:6), pp. 1399-1417.

Bandura, A. 1977. *Social Learning Theory*, Englewood Cliffs, NJ: Prentice Hall.

Belenzon, S., and Schankerman, M. 2015. "Motivation and Sorting of Human Capital in Open Innovation," *Strategic Management Journal* (36:6), pp. 795-820.

Bennett, W. L. 2003. "New Media Power," in *Contesting Media Power: Alternative Media in a Networked World*, N. Couldry and J. Curran (eds), Lanham, MD: Rowman & Littlefield, pp. 17-37.

Breschi, S., and Lissoni, F. 2009. "Mobility of Skilled Workers and Co-invention Networks: An Anatomy of Localized Knowledge Flows," *Journal of Economic Geography* (9:4), pp. 439-468.

Bird, C., Rigby, P. C., Barr, E. T., Hamilton, D. J., German, D. M., and Devanbu, P. 2009. "The Promises and Perils of Mining Git, Mining Software Repositories," in *Proceedings of 6th IEEE International Working Conference on Mining Software Repositories (MSR)*, Vancouver, Canada, May 16-17.

Blincoe, K., Sheoran, J., Goggins, S., Petakovic, E., and Damian, D. 2016. "Understanding the Popular Users: Following, Affiliation Influence and Leadership on GitHub," *Information and Software Technology* (70), pp. 30-39.

Bono, J. E., and Judge, T. A. 2004. "Personality and Transformational and Transactional Leadership: A Meta-Analysis," *Journal of Applied Psychology* (89:5), pp. 901-910.

Braunschweig, B., Dhage, N., Viera, M. J., Seaman, C., Sampath, S., and Koru, G. A. 2012. "Studying Volatility Predictors

in Open Source Software," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* Lund, Sweden, September 19-20.

Carsten, M. K., Uhl-Bien, M., West, B. J., Patera, J. L., and McGregor, R. 2010. "Exploring Social Constructions of Followership: A Qualitative Study," *The leadership Quarterly* (21:3), pp. 543-562.

Chengalur-Smith, I., Sidorova, A., and Daniel, S. 2010. "Sustainability of Free/Libre Open Source Projects: A Longitudinal Study," *Journal of the Association for Information Systems* (11:SI), pp. 657-683.

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

Collinson, D. 2006. "Rethinking Followership: A Post-Structuralist Analysis of Follower Identities," *The Leadership Quarterly* (17:2), pp. 179-189.

Crowston, K., Annabi, H., and Howison, J. 2003. "Defining Open Source Software Project Success," in *Proceedings of the 24th International Conferences of Information Systems*, Seattle, WA, December 14-17.

Crowston, K., and Howison, J. 2006. "Hierarchy and Centralization in Free and Open Source Software Team Communications," *Knowledge, Technology & Policy* (18:4), pp. 65-85.

Crowston, K., Howison, J., and Annabi, H. 2006. "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice* (11:2), pp. 123-148.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. 2012. "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, Seattle, WA, February 11-15.

Dahlander, L., and Magnusson, M. G. 2005. "Relationships Between Open Source Software Companies and Communities: Observations from Nordic Firms," *Research Policy* (34:4), pp. 481-493.

Dahlander, L., and O'Mahony, S., 2011. "Progressing to the Center: Coordinating Project Work," *Organization Science* (22:4), pp. 961-979.

Daniel, S., Agarwal, R., and Stewart, K. J. 2013. "The Effects of Diversity in Global, Distributed Collectives: A Study of Open Source Project Success," *Information Systems Research* (24:2), pp. 312-333.

Desouza, K. C., Awazu, Y., and Tiwana, A. 2006. "Four Dynamics for Bringing Use Back into Software Reuse," *communications of the ACM* (49:1), pp. 96-100.

Di Gangi, P. M., and Wasko, M. 2009. "Steal My Idea! Organizational Adoption of User Innovations from a User Innovation Community: A Case Study of Dell IdeaStorm," *Decision Support Systems* (48:1), pp. 303-312.

Dvir, T., Eden, D., Avolio, B. J., and Shamir, B. 2002. "Impact of Transformational Leadership on Follower Development and Performance: A Field Experiment," *Academy of Management Journal* (45:4), pp. 735-744.

Fershtman, C., and Gandal, N. 2011. "Direct and Indirect Knowledge Spillovers: The 'Social Network' of Open Source Projects," *The RAND Journal of Economics* (42:1), pp. 70-91.

Franke, N., and von Hippel, E. 2003. "Satisfying Heterogeneous User Needs via Innovation Toolkits: The Case of Apache Security Software," *Research Policy* (32:7), pp. 1199-1215.

Gong, Y., Huang, J. C., and Farh, J. L. 2009. "Employee Learning Orientation, Transformational Leadership, and Employee Creativity: The Mediating Role of Employee Creative Self-Efficacy," *Academy of management Journal* (52:4), pp. 765-778.

Grewal, R., Lilien, G. L., and Mallapragada, G. 2006. "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management Science* (52:7), pp. 1043-1056.

Gulati, R. 1999. "Network Location and Learning: The Influence of Network Resources and Firm Capabilities on Alliance Formation," *Strategic Management Journal* (20:5), pp. 397-420.

Haefliger, S., von Krogh, G., and Spaeth, S. 2008. "Code Reuse in Open Source Software," *Management Science* (54:1), pp. 180-193.

Hahn, J., Moon, J. Y., and Zhang, C. 2008. "Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties," *Information Systems Research* (19:3), pp. 369-391.

Hannah, S. T., and Lester, P. B. 2009. "A Multilevel Approach to Building and Leading Learning Organizations," *The Leadership Quarterly* (20:1), pp. 34-48.

Haslam, S. A., and Platow, M. J. 2001. "The Link Between Leadership and Followership: How Affirming Social Identity Translates Vision into Action," *Personality and Social Psychology Bulletin* (27:11), pp. 1469-1479.

Hinds, P. S., Gattuso, J., and Morrell, A. 2000. "Creating a Hospital-Based Nursing Research Fellowship Program for Staff Nurses," *Journal of Nursing Administrating* (30:6), pp. 317-324.

Horta, H., Veloso, F. M., and Grediaga, R. 2010. "Navel Gazing: Academic Inbreeding and Scientific Productivity," *Management Science* (56:3), pp. 414-429.

Howell, J. M., and Shamir, B. 2005. "The Role of Followers in the Charismatic Leadership Process: Relationships and Their Consequences," *Academy of Management Review* (30:1), pp. 96-112.

Ilies, R., Morgeson, F. P., and Nahrgang, J. D. 2005. "Authentic Leadership and Eudaemonic Well-Being: Understanding Leader–Follower Outcomes," *The Leadership Quarterly* (16:3), pp. 373-394.

Kogut, B., and Metiu, A. 2001. "Open Source Software Development and Distributed Innovation," *Oxford Review of Economic Policy* (17:2), pp. 248-264.

Kovarik, B. 2015. *Revolutions in Communication: Media History from Gutenberg to the Digital Age*, New York: Bloomsbury Publishing.

Lakhani, K. R., and von Hippel, E. 2003. "How Open Source Software Works: 'Free' User-to-User Assistance," *Research Policy* (32:6), pp. 923-943.

Lavie, D. 2006. "The Competitive Advantage of Interconnected Firms: An Extension of the Resource-Based View," *Academy of Management Review* (31:3), pp. 638-658.

Li, Y., Tan, C. H., and Teo, H. H. 2012. "Leadership Characteristics and Developers' Motivation in Open Source Software Development," *Information & Management* (49:5), pp. 257-267.

Loeliger, J., and McCullough, M. 2012. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, Sebastopol, CA: O'Reilly Media.

Lord, R. G., and Brown, D. J. 2003. *Leadership Processes and Follower Self-Identity*, Hove, UK: Psychology Press.

MacCormack, A., Rusnak, J., and Baldwin, C. Y. 2006. "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Management Science* (52:7), pp. 1015-1030.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions of Software Engineering Methodology* (11:3), pp. 309-346.

Napier, B. J., and Ferris, G. R. 1993. "Distance in Organizations," *Human Resource Management Review* (3:4), pp. 321-357.

O'Mahony, S., and Ferraro, F. 2007. "The Emergence of Governance in an Open Source Community," *Academy of Management Journal* (50:5), pp. 1079-1106.

Palvia, P., Pinjani, P., and Sibley, E. H. 2007. "A Profile of Information Systems Research Published in *Information & Management*," *Information & Management* (44:1), pp. 1-11.

Paulson, L. D. 2007. "Developers Shift to Dynamic Programming Languages," *Computer* (40:2), pp. 12-15.

Rainer, A., and Gale, S. 2005. "Evaluating the Quality and Quantity of Data on Open Source Software Projects," in *Proceedings of 1st International Conference on Open Source Software*, Geona, Italy.

Riggio, R. E., Chaleff, I., and Lipman-Blumen, J. (eds.). (2008). *The Art of Followership: How Great Followers Create Great Leaders and Organizations*, Hoboken, NJ: John Wiley & Sons.

Riquelme, F., and González-Cantergiani, P. 2016. "Measuring User Influence on Twitter: A Survey," *Information Processing & Management* (52:5), pp. 949-975.

Roberts, J. A., Hann, I. H., and Slaughter, S. A. 2006. "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management Science* (52:7), pp. 984-999.

Robles, G., and González-Barahona, J. M. 2012. "A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes," in *Open Source Systems: Long-Term Sustainability*, I. Hanniydam, B. Lundell, T. Mikkonen, and W. Scacchi (eds.), Berlin: Springer, pp. 1-14.

Rus, I., and Lindvall, M. 2002. "Knowledge Management in Software Engineering," *IEEE Software* (19:3), pp. 26-38.

Sawhney, M., Verona, G., and Prandelli, E. 2005. "Collaborating to Create: The Internet as a Platform for Customer Engagement in Product Innovation," *Journal of Interactive Marketing* (19:4), pp. 4-17.

Sen, R. 2007. "A Strategic Analysis of Competition Between Open Source and Proprietary Software," *Journal of Management Information Systems* (24:1), pp. 233-257.

Sen, R., Singh, S. S., and Borle, S. 2012. "Open Source Software Success: Measures and Analysis," *Decision Support Systems* (52:2), pp. 364-372.

Setia, P., Rajagopalan, B., Sambamurthy, V., and Calantone, R. 2012. "How Pperipheral Developers Contribute to Open-Source Software Development," *Information Systems Research* (23:1), pp. 144-163.

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

Singh, P. V. 2010. "The Small-World Effect: The Influence of Macro-Level Properties of Developer Collaboration Networks on Open-Source Project Success," *ACM Transactions on Software Engineering and Methodology* (20:2), Article 6.

Singh, P. V., Tan, Y., and Mookerjee, V. 2011. "Network Effects: The Influence of Structural Capital on Open Source Project Success," *MIS Quarterly* (35:4), pp. 813-829.

Skilton, P. F., and Dooley, K. J. 2010. "The Effects of Repeat Collaboration on Creative Abrasion," *Academy of Management Review* (35:1), pp. 118-134.

Sojer, M., and Henkel, J. 2010. "Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments," *Journal of the Association for Information Systems* (11:12), pp. 868-901.

Spinellis, D., and Szyperski, C. 2004. "How Is Open Source Affecting Software Development?," *IEEE Software* (21:1), pp. 28-33.

Stewart, D. 2005. "Social Status in an Open-Source Community," *American Sociological Review* (70:5), pp. 823-842.

Stewart, K. J., Ammeter, A. P., and Maruping, L. M. 2006. "Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects," *Information Systems Research* (17:2), pp. 126-144.

Uhl-Bien, M., Riggio, R. E., Lowe, K. B., and Carsten, M. K. 2014. "Followership Theory: A Review and Research Agenda," *The Leadership Quarterly* (25:1), pp. 83-104.

von Hippel, E., and von Krogh, G. 2003. "Open Source Software and the 'Private–Collective' Innovation Model: Issues for Organization Science," *Organization Science* (14:2), pp. 209-223.

von Krogh, G., Haefliger, S., and Spaeth, S. 2003. "Collective Action and Communal Resources in Open Source Software Development: The Case of Freenet," in *Proceedings of Academy of Management*, Seattle, WA, August 2003.

von Krogh, G., Haefliger, S., Spaeth, S., and Wallin, M. W. 2012. "Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development," *MIS Quarterly* (36:2), pp. 649-676.

von Krogh, G., Spaeth, S., and Haefliger, S. 2005. "Knowledge Reuse in Open Source Software: an Exploratory Study of 15 Open Source Projects," in *Proceedings of the 38th Hawaii International Conference on System Sciences*, January 3-6.

Walker, M. E., Wasserman, S., and Wellman, B. 1994. "Statistical Models for Social Support Networks," in *Advances in Social Network Analysis: Research in the Social and Behavioral Sciences* (Vol 171), S. Wasserman and J. Galaskiewicz (eds.), Thousand Oaks, CA: Sage Publication, pp. 53-53.

Zhu, K. X., and Zhou, Z. Z. 2012. "Research Note-Lock-In Strategy in Software Competition: Open-Source Software vs. Proprietary Software," *Information Systems Research* (23:2), pp. 536-545.

About the Authors

Qiqi Jiang is an assistant professor in the Department of Digitalization, Copenhagen Business School. He obtained his Ph.D. in Information Systems from City University of Hong Kong. His research interests include open innovation, marketing and data analytics, knowledge management, and electronic and mobile commerce. His work has been published or forthcoming in *MIS Quarterly*, *Journal of Management Information System*, and *Information & Management*, and others.

Chuan-Hoo Tan is an associate professor of Information Systems in the Department of Information Systems and Analytics, School of Computing, National University of Singapore. His current research interests include two streams: (1) IT design and evaluation (including human–computer interaction, digital commerce, and business analytics), and (2) IT implementation and management (including open innovation, IT design, adoption, and usage). His articles have appeared in *MIS Quarterly*, *Information Systems Research*, *Journal of Management Information Systems*, *Long Range Planning*, *Decision Support Systems*, *Annual of Operations Research,* and others.

Choon Ling Sia is a professor at the City University of Hong Kong and National Taiwan University and an AIS Fellow (2015). He received his Ph.D. from the National University of Singapore**.** His research interests include electronic commerce, social media, cross-cultural issues in information systems, knowledge management, distributed work arrangements, and computer-mediated communication. His articles have appeared in *MIS Quarterly*, *Information Systems Research*, *Journal of International Business Studies, Journal of Management Information Systems*, *ACM Transactions on Computer-Human Interaction,* and others.

Kwok-Kee Wei is the Provost's Chair Professor of Information Systems in the Department of Information Systems and Analytics, School of Computing, National University of Singapore. He is also the Dean of the School of Continuing and Lifelong Education (SCALE), National University of Singapore. He is a Fellow of the Association for Information Systems; he received the LEO Award for Lifetime Exceptional Achievement in Information Systems in 2015. His articles have appeared in top-tier information systems conferences and journals such as *MIS Quarterly*, *Information Systems Research*, *Journal of Management Information Systems*, and *Information & Management*. His current research areas include knowledge management systems, human–computer interaction, innovation adoption and management, and electronic commerce.

MIS Quarterly

RESEARCH ARTICLE

FOLLOWERSHIP IN AN OPEN-SOURCE SOFTWARE PROJECT AND ITS SIGNIFICANCE IN CODE REUSE

Qiqi Jiang
Department of Digitalization, Copenhagen Business School, Howitzvej 60,
Frederiksberg 2000 DENMARK {qj.digi@cbs.dk}

Chuan-Hoo Tan
Department of Information Systems and Analytics, National University of Singapore,
13 Computing Drive, 117417 SINGAPORE {tancho@comp.nus.edu.sg}

Choon Ling Sia
Department of Information Systems, City University of Hong Kong, Tat Chee Avenue 83, Hong Kong, SAR CHINA {iscl@cityu.edu.hk}
and Department of Information Management, National Taiwan University, Taipei City 106, Taiwan R.O.C. {iscl@cityu.edu.hk}

Kwok-Kee Wei
Department of Information Systems and Analytics, National University of Singapore,
13 Computing Drive, 117417 SINGAPORE {weikk@comp.nus.edu.sg}

Appendix

Summary of Related Studies Examining OSS Leadership

Table A1. Seminal Studies Examining Leadership in the OSS Projects or Communities

Study	Methods (and Data)	Key Findings and Argument in Leadership
Fielding (1999)	Case study: Case study of Apache project.	Different from an authoritative leadership in most OSS projects, the author unveiled a shared leadership mechanism in Apache project in which a small group of controlling members uses a relatively democratic decision process.
Lerner and Tiróle (2002)	Conceptual and literature reviews.	The performance and success of an OSS project are dependent on the presence of a credible leader or leadership. They suggested that a strong centralization of authority and leadership characterizes OSS projects, such as Linux.
Fleming and Waguespack (2007)	Quantitative investigation: Archival data from Internet Engineering Task Force documents.	This study identified and discussed two imperative capabilities of OSS leadership: strong technical contribution and binding multiple communities.

Table A1. Seminal Studies Examining Leadership in the OSS Projects or Communities

Study	Methods (and Data)	Key Findings and Argument in Leadership
O'Mahony and Ferraro (2007)	Mixed method. Qualitative investigation: Interview with Debian contributors. Quantitative investigation: Regression on the variables constructed from multisourced archival data including project directory, developer database, bug-tracking database, package popularity database, and identity authentication database.	This work discussed the role of leadership in governing an OSS project. The authority of leadership can be highlighted in the bureaucratic mechanism, which increases the management efficiency. The leaders can construct the democratic governance mechanism to represent the community's interests and adapt with members' evolving interpretation of leadership.
Giuri et al. (2008)	Quantitative investigation: Econometric analysis with three-year archival data from Sourceforge.net.	OSS project leader with diversified skill set can coordinate the contribution from various participants and motivate such participants. The presence of leadership is associated with the degree of modularity of the development process.
Li et al. (2012)	Quantitative investigation: Structural equation modeling with survey data contributed by 187 developers from Sourceforge.net.	Leader can use transformational leadership and active management style to positively stimulate developers' intrinsic and extrinsic motivations, respectively.
Faraj et al. (2015)*	Quantitative investigation: Archival data collected from three different Usenet newsgroups.	This work argued that the leadership in community-based production groups can be identified by (1) extended contributed knowledge; (2) sociability, that is, facilitating communication among online participants; and (3) structural social capital, that is, positions in the online social network.
Johnson et al. (2015)	Quantitative investigation: Using survey to identify the leader of each community; applying Natural Language Processing package to extract primary variables from archival data.	This work argued that the following factors are instrumental in forming leadership: (1) occupying a formally authoritative role in community-based production groups, (2) being better positioned in the communication network, and (3) using unique patterns of language.

*Although Usernet newsgroups are not typical OSS project groups, the nature of these groups in this study, namely, object-oriented programming, databases, and C++ programming, are similar to those of OSS. The authors also cited OSS studies when they developed their theoretical arguments. Thus, we included this work in this table.

Post Hoc Analyses

In the first set of *post hoc* analyses, the data were subjected to two conventional robustness checks, namely, varying the operationalization of the dependent variable and varying the regression analysis models. We varied the dependent-variable operationalization from $AvgDiff\_CodeReuse_i$ (i.e., the average difference in code reuse among leader i's OSS projects at T_2 and T_1) to $AvgRatio\_CodeReuse_i$. This variable signifies the average change (as a ratio) in the extent of code reuse for leader i's projects at T_2 compared with that at T_1. A high value of $AvgRatio\_CodeReuse_i$ indicates a great extent of code reuse among that leader's OSS projects. Multilevel regression was applied to estimate the coefficients. Two additional regression models were used to estimate the relationship between the predictors and the focal dependent variables, $AvgDiff\_CodeReuse_i$. Specifically, we applied this variable to reconcile the effect of nuisance parameters (Model 2 in Table A2) and estimate the influence of key predictors and the related control variables for $AvgDiff\_CodeReuse_i$ with random-intercept model (Model 3 in Table A2) (McCulloch and Neuhaus 2001). Table A2 presents the summary of results. The results show significant consistency with the earlier findings obtained in the main testing.

Table A2. First *Post Hoc* Analysis

Dependent Variables	$AvgRatio\_CodeReuse_i$	$AvgDiff\_CodeReuse_i$	
	Model 1	**Model 2**	**Model 3**
$Developer\_followership_{ijt1}$	−0.088** (0.039)	−0.197** (0.085)	−0.200** (0.085)
$Observer\_followership_{it1}$	0.061*** (0.008)	0.221*** (0.018)	0.211*** (0.018)
$Project\_age_{jit1}$	−0.017 (0.031)	0.076 (0.069)	0.053 (0.071)
$Leader\_age_{ijt1}$	−0.756*** (0.104)	−1.515*** (0.253)	−1.544*** (0.241)
$Leader\_following_{ijt1}$	0.004 (0.009)	−0.062*** (0.018)	−0.062*** (0.018)
$Project\_size_{jit1}$	0.002 (0.009)	−0.012 (0.019)	−0.021 (0.020)
$Project\_knowledge_{jit1}$	−0.128*** (0.039)	−0.243*** (0.090)	−0.249*** (0.088)
$Cumulative\_Version_{jit1}$	−0.012 (0.008)	−0.024 (0.019)	−0.010 (0.019)
$Ratio_{jit1}$	−0.010 (0.014)	0.096** (0.040)	0.103*** (0.036)
$Recency_{jit1}$	−0.036 (0.012)	−0.035 (0.025)	−0.025 (0.025)
$Project\_observer_{jit1}$	0.086*** (0.011)	0.144*** (0.026)	0.183*** (0.027)
$Language\_Popularity_{jit1}$	−0.416 (0.430)	0.325 (0.905)	−0.174 (0.946)
$Language\_Media\_Coverage_{jit1}$	−0.006 (0.005)	0.005 (0.011)	0.005 (0.012)
Intercept	5.774*** (0.859)	11.388*** (2.053)	11.613*** (1.951)
Deviance (-2 × log-likelihood)	1000.527	2398.375	2356.345
Akaike's information criterion	1082.527	2484.375	2438.345
Bayesian information criterion	1284.557	2693.654	2637.889

*p ≤ 0.1; **p ≤ 0.05; ***p ≤ 0.01; figures are displayed in the format of "Coefficient (Std. Err.)"

Note: For Model 1, we replaced the dependent variable by $AvgRatio\_CodeReuse_i$ (average change as a ratio in the extent of code reuse for leader i's projects at T2 compared with T1). For Model 2, we applied restricted maximum likelihood estimation for reconciling the effect of nuisance parameters. For Model 3, we applied the random-intercept model for estimating the coefficients. The results from the three models are consistent with those in Table 4 in the main body. To easily interpret the results, we scaled down $Project\_age_{jit1}$, $Ratio_{jit1}$, and $Recency_{jit1}$ by factors of 1000, 100, and 100, respectively. The programming languages, denoted by $Programming\_Language_{ji}$, and project category, denoted by $Project\_Category_{ji}$, were created as dummy variables and estimated in the regression model but are not reported for brevity.

In the second set of *post hoc* analyses, we segmented the OSS projects into affiliated categories that could have highly diverse complexity in OSS development (Au et al. 2009; Sen 2007). The dataset contained three project categories: Software Framework, Application Software, and Documentation File. Two interesting findings can be drawn from the results in Table A3. First, a large number of followers play the role of observer; specifically, a close observer of the leader can enhance code reuse across the types of OSS projects. Second, given that the Software Framework (Model 1 in Table A3) and Application Software (Model 2 in Table A3) types are hosted on GitHub, developers have to constantly contribute. Therefore, the developer–follower relationship has a significant positive effect on code reuse. Miscellaneous OSS textual files, such as Cascading Style Sheet files for websites or unified configuration files for software, were included in Model 3 in Table A3. In contrast to the previous two categories (whose operations heavily rely on developers), the OSS projects in Documentation File can be used extensively. However, developers in this study show minimal interest in such relatively insignificant projects. Consequently, the results in this third case show that $Developer\_followership_{ijt1}$ does not influence the extent of code reuse.

Table A3. Second *Post Hoc* Analysis

Dependent Variables	$AvgDiff\_CodeReuse_i$		
	Model 1	**Model 2**	**Model 3**
$Developer\_followership_{ijt1}$	−0.193* (0.119)	−0.357** (0.147)	0.121 (0.172)
$Observer\_followership_{it1}$	0.244*** (0.026)	0.249*** (0.031)	0.210*** (0.035)
$Project\_age_{jit1}$	0.050 (0.103)	−0.002 (0.121)	−0.011 (0.128)
$Leader\_age_{ijt1}$	−4.569*** (0.425)	−0.892* (0.529)	−0.080 (0.335)
$Leader\_following_{ijt1}$	−0.069*** (0.025)	−0.046 (0.032)	−0.058 (0.036)
$Project\_size_{jit1}$	−0.030 (0.026)	−0.006 (0.032)	−0.008 (0.051)
$Project\_knowledge_{jit1}$	−0.147 (0.112)	−0.650*** (0.183)	−0.179 (0.175)
$Cumulative\_Version_{jit1}$	−0.010 (0.026)	−0.063* (0.034)	0.044 (0.038)
$Ratio_{jit1}$	0.014 (0.054)	0.231* (0.119)	0.196*** (0.050)
$Recency_{jit1}$	−0.025 (0.035)	0.006 (0.042)	−0.046 (0.056)
$Project\_observer_{jit1}$	0.197*** (0.037)	0.170*** (0.052)	0.078 (0.052)
$Language\_Popularity_{jit1}$	0.209 (1.352)	−1.936 (1.648)	−0.790 (1.802)
$Language\_Media\_Coverage_{jit1}$	−0.004 (0.015)	0.035 (0.021)	0.012 (0.024)
Intercept	34.075*** (3.196)	6.652* (3.937)	0.732 (2.475)
Deviance (−2 × log-likelihood)	1123.358	545.363	541.493
Akaike's information criterion	1193.358	609.5363	605.149
Bayesian information criterion	1339.441	722.6047	714.748

*p ≤ 0.1; **p ≤ 0.05; ***p ≤ 0.01; figures are displayed in the format of "Coefficient (Std. Err.)"

Note: We separated the samples into three sub-samples by the software categories, namely, Software Framework (Model 1), Application Software (Model 2), and Documentation Files (Model 3). To easily interpret the results, we scaled down $Project\_age_{jit1}$, $Ratio_{jit1}$, and $Recency_{jit1}$ by factors of 1000, 100, and 100, respectively. The programming languages, denoted by $Programming\_Language_{ji}$, were created as dummy variables and estimated in the regression model but are not reported for brevity.

The third set of *post hoc* analyses was conducted to check for potential multicollinearity problems. In particular, the analysis used the ordinary least squares model. In this step, we also calculated VIF and condition index (CI) values to assess potential multicollinearity problems. The multicollinearity in the stated results should not be a problem because no VIF calculations exceed 5.0, and the CI value is 24.838 (Ho and Richardson 2013). The robustness of the model was also validated and demonstrated by adding the control variables into the main model by using a stepwise approach. Due to the page limitation, the details of estimated results are available from the authors upon request.

The fourth set of *post hoc* analyses was conducted to rule out an alternative explanation for "follower inbreeding" among certain OSS projects. The results discussed in the preceding paragraphs indicate that the followership conceived through prior collaboration is not conducive to code reuse. However, these findings may be the result of an alternative explanation: enhanced code reuse may not be attributed to the open developer collaboration but can be due to the scarce participation of certain developers in other OSS projects. Their lack of enthusiasm may be caused by extreme loyalty to specific leaders. Thus, the extant projects with tight developer–follower relationships were split into two

subgroups, namely, those who worked only on a leader's project and those who worked on multiple projects with different leaders. If no difference exists in code reuse between the two groups, then the concern of "follower inbreeding" can be minimized.

Four steps were taken to test this possibility at the project level. First, a project-level dependent variable, $Diff\_CodeReuse_j$, was constructed. This variable denotes the difference in the extent of code reuse of the OSS project j managed by the same project leader between T_1 and T_2. Next, a new binary variable, $Single\_leader_{jtl}$, was constructed. This variable indicates whether the focal OSS project j is administered by a single leader (a value of 1 indicates that the project was administered by a single leader). Third, using the descriptive statistics of $Developer\_followership_{ijtl}$ (mean = 0.320 and S.D. = 0.371) presented in Table 2, the OSS projects with an extent of collaborative followership greater than 0.3 (a similar value to the sum of the mean and S.D.) were labeled as projects with a tight developer–follower relationship. A high value of $Developer\_followership_{ijtl}$ indicates a tight relationship. Finally, the control variables were transformed into the project level[1] and entered as $Single\_leader_{itl}$ into the OLS regression model. Although the selected OSS projects can be statistically accepted as the OSS projects with high developer followers ($Developer\_followership_{ijtl}$ is greater than 0.3), the regression models were consistently run with different samples through a repeated increase in the value of $Developer\_followership_{ijtl}$ from 0.3 to 0.9 by intervals of 0.1. The estimated coefficient of $Single\_leader_{jtl}$ is insignificant under different extent of collaborative followership. Thus, we can conclude that no difference exists in code reuse between the two groups. Due to the page limitation, the details of estimated results are available from the authors upon request.

Additional Investigations: Survey and Focus Groups

We conducted two *post hoc* investigations in collaboration with an IT research agency to validate our arguments regarding code reuse and the underlying role of learning. The first *post hoc* investigation included a survey, which had 164 complete responses from GitHub-registered members. The second *post hoc* investigation included focus groups of 24 GitHub-registered members.[2] Three focus group were held with eight GitHub-registered members in each (for approximately 90–120 minutes each).[3] The qualitative data[4] extracted from the focus groups enriched the survey findings. Below we present results from both investigations.

We grounded the survey in the social learning framework (Bandura 1977). The aim was to test our conjecture that code reuse (as measured in the survey as the intention to reuse the codes) can be contributed by followers who are learning from their leaders. Figure A1 presents the model. Intention to code reuse is the outcome variable. We included several control variables related to OSS values and beliefs, namely, the perception of importance to (1) code reuse, (2) sharing mentality, and (3) OSS team diversity, tenure in GitHub, OSS usage experience, gender, and age. Table A5 shows the operalization of each construct. We used SmartPLS 2.0.M3 to test the model. Table A4 reports the estimated path coefficients, and the results for construct validity and reliability.

[1] $Leader\_age_{ijtl}$ (age of leader i) and $Leader\_following_{ijtl}$ (the number of people followed by leader i) were excluded because the two variables were typically leader-level covariates.

[2] All interviewees first introduced themselves and shared their OSS experience. They were then asked about their general understanding of OSS beliefs and values: two key components of OSS ideology, code reuse in software development (especially for the OSS project), and their experience as the followers (as developers or observers) in one or more OSS projects. The group interviews were recorded in an audio–visual format. Each interviewee received a compensation of US$50 for his or her time.

[3] A total of 24 interviewees were involved, including active OSS participants with numerous experiences in OSS development and relatively passive OSS participants who mostly lurk in the community. These 24 interviewees (14 males and 10 females) were arranged into three focus groups to hold further discussions. The average programming experience (in the real project) of these respondents is approximately 5 years. Their programming language ranges from objective-oriented languages (e.g., Java, Objective-C, or C#) to scripting language (e.g., PHP, JavaScript, Python, or Ruby).

[4] Two independent assistants were recruited to transcribe the protocols of the focus groups. Two coders were then recruited to code the protocols following standard procedures (Someren et al. 1994). Their coding was very similar (Cohen's kappa = 0.88, $p < 0.010$), indicating high inter-coder reliability (Cohen and Reed 2006). Respondents were first asked to describe their understanding of OSS, general thoughts on code reuse, and personal opinions about the follower–leader relationship. Approximately 12,000 common words in the answers of each focus group discussion were found. Table A6 summarizes some representative codes and descriptions.

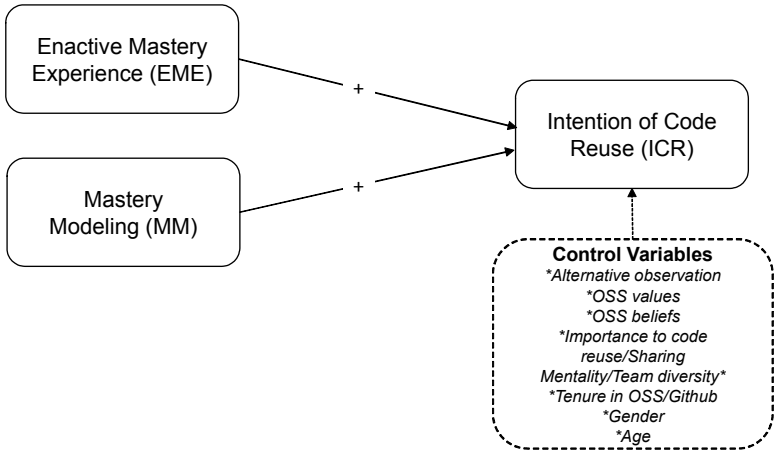

Figure A1. Conceptual Model of the Follow-up Survey

Table A4. Results of the *Post Hoc* Survey Investigation

	Estimated Path Coefficient	AVE	Composite Reliability	Cronbach's Alpha
MM	0.15**	0.638	0.876	0.815
EME	0.24***	0.609	0.817	0.701

R square = 0.59; *p value ≤ 0.1; **p value ≤ 0.05; ***p value ≤ 0.01

Table A4 shows that mastery modeling (MM) and enactive mastery experience (EME) significantly relate to the intention to code reuse. This finding provides empirical support for our suggestion that learning from OSS followers can explain the relationship between followership and code reuse. Specifically, developers as followers can obtain an in-depth understanding of the OSS projects. This situation renders them competent in applying the source codes or modular code components from their endeavoring projects in their other OSS projects. As one interviewee (FG-3-E[5]) stated, "*By working closely with the (OSS project) leader, I can have a clear image of the modular architecture of that focal project. If I need, I can directly reuse the modules or source codes (from certain modules) for other OSS projects that I work on. This really saves me a lot of effort; namely I can avoid reinventing the wheel.*"

In addition, regarding EMEs, two interesting findings were also unveiled. First, EME is developed through interaction with the leaders. As one interviewee (FG-3-H) stated, "*I received the response from the leader regarding my editing in the source codes. The approval (of my editing) conveyed the message of endorsement (of my work). Even my changes were rejected; I mostly received the comments (posted by leaders) indicating the reasons for rejection. Such interactive activities really improved my coding capability. This feedback is a personal instructor.*" Second, EME can be cultivated through collaborative diversity, as noted by another interviewee (FG-2-C): "*I can learn more through working with different people. There are breakthrough results from diversity. I don't think it is a good idea to constantly subordinate myself to the same people like in my daily employment.*"

The observers as followers are updated on the activities of their interested (OSS project) leaders, which helps them acquire miscellaneous information about software development and project management. Consequently, these observers will apply this information to the other projects. For example, an interviewee (FG-2-G) indicated, "*I follow some skilled leaders of OSS projects in GitHub to learn from them. This provides me [with] great opportunities to learn the codes they wrote and the way they managed the projects. Sometimes, I seek the codes I need from these people's activities.*" Such an opinion is also echoed by another interviewee (FG-1-A): "*Reputable people are followed to gain trendy information.*"

Interestingly, the learning cost in MM is relatively lower than that in EME. The observers can concurrently follow as many of the OSS project leaders as they want to expand their scope of knowledge. As stated by one interviewee (FG-1-C), "*I just follow those who have interesting projects in GitHub. There is no cost for me to do so, and I may get important information for my own projects from their updates and*

[5]FG-3-E denotes Focus Group 3, interviewee E.

activities." Theoretically, having observers does not cost the OSS project leaders. As expressed by one interviewee (FG-3-C), "*the following function in GitHub is extremely convenient and provides mutual benefits. We (observers) can easily get project-related information by following its leader, and that leader can also rely on (leveraging) us to promote her/his project.*" This comment provides descriptive explanations for research conjecture 2.

We gain additional interesting findings from the focus group interviews. As expected, code reuse is widely prevalent in developing an OSS project. Most interviewees agree with the following statement made by an interviewee (FG-2-E): "*I directly copy the codes from one project on which I work and paste them into another one (that I work for) …. No need to reinvent the wheel.*" This opinion is also echoed by another interviewee (FG-2-D): "*GitHub is like a bazaar where I can reuse the codes from other (developing) projects*." We find that the OSS participants hold a strong sharing mentality when they are subsequently asked to express their opinions in the activities of code reuse. An exemplary quote (from interviewee FG-3-D), "*The OSS is conducive to reducing the load of creating software from scratch. We can find needed codes for the existing projects, and we are not penalized for doing so*," adequately corroborates the assumption of the sharing mentality in OSS values and beliefs.

Table A5. Operationalization of the Constructs

Constructs and Sources	Items	Scale
Dependent Variable		
Intention to code reuse (Fishbein and Ajzen 1975; Sheppard et al. 1988)	(1) I intend to reuse the code learned from one OSS project leader in another OSS project. (2) I plan to continue reusing the code learned from one OSS project leader in another OSS project. (3) I plan to routinely reuse the code learned from one OSS project leader in another OSS project. (4) I predict that I will reuse the code learned from one OSS project leader in another OSS project.	Seven-point Likert scale (1 = strongly disagree; 4 = neutral; 7 = strongly agree)
Independent Variables		
EME (Self-developed)	(1) By following a particular person and contributing to her/his OSS project in GitHub, (2) I frequently learn from her/him. (3) The experience I learn from her/him is not valuable for me (reverse coded). (4) Co-working with her/him does not help me develop my experiences (reverse coded).	Seven-point Likert scale (1 = strongly disagree; 4 = neutral; 7 = strongly agree)
MM (Bandura 1977)	(1) I am able to enrich my knowledge by noting the information from the OSS project leaders I follow. (2) I have developed confidence in my ability to develop my own works by observing the information from the OSS project leaders I follow. (3) Gathering information from the OSS project leaders I follow has taught me how to develop my own works. (4) I have learned how to develop my own works better by monitoring the OSS project leaders I follow.	Seven-point Likert scale (1 = strongly disagree; 4 = neutral; 7 = strongly agree)

Table A5. Operationalization of the Constructs (Continued)

Constructs and Sources	Items	Scale
Control Variables		
Attentive observation (Yi and Davis 2003)	(1) I have paid close attention to the news or information on GitHub. (2) The news or information gets my attention. (3) I am able to take notice of the news or information on GitHub. (4) When using GitHub, I am absorbed by the news or information presented.	Seven-point Likert scale (1 = strongly disagree; 4 = neutral; 7= strongly agree)
OSS values (Stewart and Gosain 2006)	(1) I value sharing knowledge. (2) I believe in helping others. (3) I place great value on technical knowledge. (4) I am driven by a desire to learn new things. (5) I think cooperation is important. (6) I value the reputation gained from participating in open-source projects.	Seven-point Likert scale (1 = strongly disagree; 4 = neutral; 7= strongly agree)
OSS belief (Stewart and Gosain 2006)	(1) I believe that the best code wins out in the end. (2) I believe free software is better than commercial software. (3) I think information should be free. (4) I believe that with enough people working on a project, any bug can be quickly found and fixed. (5) I believe that you only become a hacker when others call you a hacker.	Seven-point Likert scale (1= strongly disagree; 4 = neutral; 7 = strongly agree)
Perceived importance of code reuse/sharing mentality/team diversity (Self-developed)	(1) Reusing code is important for OSS projects. (2) Sharing is the core in OSS. (3) Cooperating with stable people across OSS projects is preferred.	Seven-point Likert scale (1 = strongly disagree; 4 = neutral; 7 = strongly agree)
Tenure in GitHub	How long have you used GitHub?	0: Less than 1 year 1: 1-3 years 2: 3-5 years 3: More than 5 years
OSS usage experiences	How long have you used the OSS products?	0: Less than 1 year 1: 1-3 years 2: 3-5 years 3: 5-7 years 4: More than 7 years
Gender	Are you _____?	0: Female, 1: Male
Age	What is your age?	0: Younger than 20 years 1: 20 years to 24 years 2: 25 years to 29 years 3: 30 years to 34 years 4: 35 years to 39 years 5: 40 years to 44 years 6: 45 years to 49 years 7: 50 years and over

Table A6. Coding			
	Codename	**Meaning**	**Percentage**
First Topic: Understanding Open-Source Software			
Word count: 10,288 Code count: 223	1-FD	Free distribution, including the reuse and redistribution of source code	32%
	1-KS	Knowledge sharing	29%
	1-SM	Social movement	15%
	1-CS	Cost-free software	13%
	1-IP	Intellectual property and licenses	10%
Second Topic: Thoughts on Code Reuse			
Word count: 7,860 Code count: 168	2-SE	Save efforts (by reusing the existing code)	37.5%
	2-FC	Efficient software development completion	32%
	2-QI	Improved quality of the reused code	20%
	2-LB	Learning through reusing (existing code)	9%
Third Topic: Dynamics in the Follower–Leader Relationship			
Word count: 9,148 Code count: 198	3-DV	Diversity of collaborators	31%
	3-RC	Reduction in communication/coordination cost	28%
	3-SK	Seeking qualified and updated knowledge	27%
	3-AL	Acquiring legitimacy	13%

References

Au, Y. A., Carpenter, D., Chen, X., and Clark, J. G. 2009. "Virtual Organizational Learning in Open Source Software Development Projects," *Information & Management* (46:1), pp. 9-5.

Bandura, A. 1977. *Social Learning Theory*, Englewood Cliffs, NJ: Prentice Hall.

Cohen, J.B., and Reed, A. 2006. "A Multiple Pathway Anchoring and Adjustment (MPAA) Model of Attitude Generation and Recruitment," *Journal of Consumer Research*, (33:1), pp. 1-15.

Faraj, S., Kudaravalli, S., and Wasko, M. 2015. "Leading Collaboration in Online Communities," *MIS Quarterly* (39:2), pp. 393-411.

Fielding, R. T. 1999. "Shared Leadership in the Apache Project," *Communications of the ACM* (42:4), pp. 42-43.

Fishbein, M., and Ajzen, I. 1975. *Belief, Attitude, Intention and Behavior: An Introduction to Theory and Research*, Reading, MA: Addison-Wesley.

Fleming, L., and Waguespack, D. M. 2007. "Brokerage, Boundary Spanning, and Leadership in Open Innovation Communities," *Organization Science* (18:2), pp. 165-180.

Giuri, P., Rullani, F., and Torrisi, S. 2008. "Explaining Leadership in Virtual Teams: The Case of Open Source Software," *Information Economics and Policy* (20:4), pp. 305-315.

Ho, S. Y., and Richardson, A. 2013. "Trust and Distrust in Open Source Software Development," *The Journal of Computer Information Systems* (54:1), pp. 84-93.

Johnson, S. L., Safadi, H., and Faraj, S. 2015. "The Emergence of Online Community Leadership," *Information Systems Research* (26:1), pp. 165-187.

Lerner, J., and Tirole, J. 2002. "Some Simple Economics of Open Source," *The Journal of Industrial Economics* (50:2), pp. 197-234.

Li, Y., Tan, C. H., and Teo, H. H. 2012. "Leadership Characteristics and Developers' Motivation in Open Source Software Development," *Information & Management* (49:5), pp. 257-267.

O'Mahony, S., and Ferraro, F. 2007. "The Emergence of Governance in an Open Source Community," *Academy of Management Journal* (50:5), pp. 1079-1106.

McCulloch, C. E., and Neuhaus, J. M. 2001. *Generalized Linear Mixed Models*, Chichester, UK: John Wiley & Sons, Ltd.

Sen, R. 2007. "A Strategic Analysis of Competition Between Open Source and Proprietary Software," *Journal of Management Information Systems* (24:1), pp. 233-257.

Sheppard, B. H., Hartwick, J., and Warshaw, P. R. 1988. "The Theory of Reasoned Action: A Meta-Analysis of Past Research with Recommendations for Modifications and Future Research," *Journal of Consumer Research* (15:3), pp. 325-343.

Someren, M. V., Barnard, Y. F., and Sandberg, J. A. 1994. *The Think Aloud Method: A Practical Approach to Modeling Cognitive Processes*, Waltham, MA: Academic Press.

Stewart, K. J., and Gosain, S. 2006. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *MIS Quarterly* (30:2), pp. 291-314.

Yi, M. Y., and Davis, F. D. 2003. "Developing and Validating an Observational Learning Model of Computer Software Training and Skill Acquisition," *Information Systems Research* (14:2), pp. 146-169.