Journal of the Association for Information Systems
JAIS

Special Issue

Following the Sun: Temporal Dispersion and Performance in Open Source Software Project Teams

Jorge A. Colazo
Business Administration
Trinity University
jcolazo@trinity.edu

Yulin Fang
Information Systems
City University of Hong Kong
ylfang@cityu.edu.hk

Abstract

Dispersion in working teams has been addressed by extant research mostly in terms of the physical distance that separates team members. Recently, the focus has shifted toward an examination of a newer construct –temporal dispersion (TD). The study of TD so far has been constrained mostly to conceptual work. This study furthers the understanding of TD through an empirical investigation of its relationship with open source software (OSS) team performance. In this paper, hypotheses are developed based on coordination theory, and analyses are performed using data collected from multiple archival sources comprising 100 OSS development teams. Results indicate that TD positively affects development speed and quality and that software complexity moderates the relation between TD and software quality. Theoretical and practical implications are discussed.

Keywords: *Open source, virtual teams, temporal dispersion, coordination theory, team performance*

* Michael Wade and Kevin Crowston were the accepting senior editors. This article was submitted on October 15, 2009 and went through two revisions.
** Both authors contributed equally to the paper.

Following the Sun: Temporal Dispersion and Performance in Open Source Software Project Teams

1. Introduction

Open source software (OSS), with roots in the free software movement of the early 1970s, has become increasingly important in recent years. The OSS movement generated an alternative software development model that attracted much academic and corporate attention (Sen, 2007, Stewart et al., 2006), becoming not only mainstream but also commercially viable, in a phenomenon called "OSS 2.0" (Fitzgerald, 2006). There are a burgeoning number of OSS initiatives: More than 230,000 projects and two million users are registered on sourceforge.net, the world's largest OSS repository, representing an increase of more than 100% from 2006 to 2009.

Many OSS projects have achieved great adoption success. For example, most web pages are delivered through the Apache server (Netcraft, 2009). MySQL has long become a major player in the database sector (Gartner, 2006). Furthermore, 60% of the largest companies in North America are implementing OSS applications, with half that percentage using them even for their mission-critical tasks (Schadler, 2004). More recently, the adoption of OSS has been identified as a top priority for software development professionals (Hammon et al., 2009).

A key to the success of OSS projects lies in their ability to foster a sustainable level of coding activity from volunteer developers (Fang and Neufeld, 2009, Gonzalez Barahona et al., 1999, Qureshi and Fang, 2009, 2011). However, as many as 80% of OSS projects fail because they cannot maintain a healthy level of activity (Hermann, 2005, Hertel et al., 2003). The observed variability in coding activity has naturally raised the question of what factors influence OSS development performance.

Two streams of OSS research have studied factors influencing development activities. While one stream focuses on understanding the motives behind individual developers' participation in OSS projects (Franke and von Hippel, 2003, Hertel et al., 2003, Ke and Zhang, 2009, Ke and Zhang, 2010, Lakhani and Wolf, 2005, Roberts et al., 2006, Von Hippel, 2001), the other focuses on identifying project characteristics instrumental to the various dimensions of OSS success. Project-level factors identified in prior research included project tenure, team size, types of software, programming language (Crowston and Scozzi, 2002), project sponsorship (Shah, 2006, Stewart et al., 2006), software licensing (Colazo and Fang, 2009, Stewart et al., 2006, Stewart and Ammeter, 2002, Stewart et al., 2005), codebase architecture (Baldwin and Clark, 2006), and project network structure (Colazo, 2010, Grewal et al., 2006).

While these studies have contributed to understanding the success of OSS projects, they have not considered an important project team structural factor: the team's temporal dispersion (TD), briefly defined here as the extent to which the working hours of team members differ (see a detailed discussion in the Theoretical Background section).

OSS project teams are considered a type of global virtual team (Crowston and Scozzi, 2002, Gallivan, 2001, Markus et al., 2000); one in which dispersed volunteer developers work together to produce software using internet-based collaboration tools. OSS project teams are dispersed not only spatially but also temporally. OSS project members' hours depend not only on the specific time zones where the members are located but also on their preferred working times. This study aims to address the following research question: How does temporal dispersion affect OSS project performance?

This research question is important for a number of reasons. First, although numerous studies have explored OSS success (Colazo et al., 2005, Crowston et al., 2003, Crowston et al., 2004, Raymond and Trader, 1999, Stewart, 2004, Stewart et al., 2005), the influence of TD on OSS project performance has not yet been researched, and the present paper breaks ground in this respect.

Second, TD is a major actionable configuration factor for project leaders, particularly in commercial or corporate software development projects, where the temporal allocation of team members could be more easily dictated. For instance, some software firms have considered a "Follow the Sun" approach

by purposely structuring globally distributed teams across different time zones to maximize coverage and, supposedly, development speed (Espinosa and Carmel, 2003). However, to date there is no rigorous empirical support for this practice, and this paper delves into the issue, which is not only academically interesting but also practically substantive.

Third, the answer to this research question in the context of OSS projects can pose important implications for the larger phenomenon of virtual teams. The current literature on virtual teams provides mixed views on the presumed influence of TD. While some researchers posit that TD can make virtual team coordination more difficult (Cramton, 2001, Hinds and Bailey, 2003, O'Leary and Cummings, 2007, Warkentin et al., 1997), others argue that some temporally dispersed teams can achieve remarkable performance (Massey et al., 2003, Yamauchi et al., 2000). However, most studies either only assume the presence of TD or dichotomously compare dispersed teams with co-located teams without consideration for the different degrees of TD teams might exhibit (O'Leary and Cummins, 2007). By focusing on TD and subsequently examining how it affects OSS team performance, this study provides insights that help reconcile mixed findings in the virtual team literature.

Drawing upon the virtual team literature and coordination theory (Malone and Crowston, 1994), this study argues that in the context of OSS development projects, TD positively affects both development speed and the quality of the code generated. This study also hypothesizes that these relationships are moderated by the complexity of the software being developed. The empirical analysis of the hypotheses is based on data collected from multiple archival sources comprising 100 OSS project teams.

2. Theoretical background

2.1 Defining temporal dispersion

Dispersion in software development teams, particularly in OSS project teams, has been examined through different lenses, for instance, dispersion in contributed effort (Olivera et al., 2008, Ortega et al., 2007), in permanence within the team (Colazo and Fang, 2009, Espinosa, 2003, Robles and Gonzalez Barahona, 2006), and more often at a distance between team members (i.e., spatial dispersion) (Cramton, 2001, Majchrzak et al., 2000, Manzevski and Chudoba, 2000, Neufeld et al. 2010, Townsend et al., 1998).

Although TD is implicit in the description of virtual teams in most studies (Cramton, 2001, Majchrzak et al., 2000, Manzevski and Chudoba, 2000, Townsend et al., 1998), the majority of the extant empirical research has loosely operationalized it by either dichotomizing dispersed teams against co-located ones or confounding TD with spatial or geographic dispersion. TD as a distinctly unique construct has not been explicitly addressed until recently (O'Leary and Cummings, 2007).

The limited existing literature, in general, has approached the understanding of TD in two ways First, TD has been assimilated to the variation in the time zones where team members are located. Time zones serve as a fairly stable, standardized, and recognizable descriptor of TD (O'Leary and Cummings, 2007). However, time zone dispersion by definition coexists with spatial dispersion, making it difficult to separate the effects. Also, there may or may not be TD even with team members working in the same time zone. For example, consider the members of a team working a typical 9-to-5 workday but disseminated in three distant cities: Toronto, New York, and Quito. These three locations are far apart but within the same universal time zone (in this case, UTC-5), and there is no TD. Conversely, consider three 8-hour shifts worked in the same location. In this example, there is no spatial dispersion but there is TD. Using time zone variation alone to measure TD would misrepresent either case. The decision of which time zone applies to a certain country or region within a country can also be affected by government policies, such as the implementation of daylight savings time. In addition, time zones are, by definition, separated by an integer number of hours, artificially fixing the granularity of the analysis.

Second, TD has been discussed as the variation in the work hours between team members (Griffith et al., 2003, Kirkman and Mathieu, 2005, Knoll, 2000). This indicator accommodates socio-cultural variations in work hours and days. It can also account for situations in which team members are located at the same site but work different shifts or even flexible hours. In this description, TD does not necessarily coexist with spatial dispersion, and yields to a more fine-grained observation of a range of TD under which a given team operates, for instance when people work shifts or when starting work at different times, whether or not the team members work in the same location. Under this approach, the working times should always be measured in a location-independent unit, such as Universal Time Coordinated (UTC).

In our particular case of OSS, the latter scenario (i.e., the variation in actual work hours) presents a more adequate approach for the following reasons. First, most OSS developers are volunteers with very flexible work hours at their disposal (Dempsey, 2002). They do not normally adhere to regular "office" hours and even their temporal work patterns may change from day to day or over longer periods of time. Thus, those who sit in the same time zone may still work at different times, showing a substantive degree of TD. Second, the time zone approach to TD is most suitable when teams have both stable geographic boundaries and membership (O'Leary and Cummings, 2007). OSS project membership is highly fluid, and thus produces a constant renewal and variation of work time information as project membership changes. Third, as explained earlier, while time zone differences, by definition, correlate with spatial dispersion, considering actual work hours alleviates the potential confounding effect of spatial dispersion. Finally, in terms of empirical feasibility, data on developers' actual work hours are much more tractable in OSS projects than developers' time zone information (see the method section for details).

Taken together, these arguments support the adoption of the variation of developers' actual work hours as the working definition for TD in OSS projects. However, it is noteworthy that, in our definition, we do not account for the emergence of TD due to new members joining or leaving the project in different generations of developers, nor do we account for the dispersion in effort put into the project or the variance in individual developer productivity.

2.2 Effect of temporal dispersion

While some studies position TD as an important dimension of virtual teams with notable performance implications along with spatial dispersion (Griffith et al., 2003, Martins et al., 2004), others take one step further by arguing that spatial dispersion is an outdated descriptor of virtual teams because co-located teams can also exhibit the behavior typical of virtual teams (Kirkman and Mathieu, 2005). Instead, they propose that TD, manifested as the degree of asynchronous communication, should be a defining characteristic of virtual teams. Despite the ongoing discussion regarding the role of TD, prior research presents largely mixed views on the effects of TD and spatial dispersion on virtual team performance (O'Leary and Cummings, 2007).

Following O'Leary and Cummings (2007), this study posits that TD and spatial dispersion are both important dimensions characterizing virtual teams, but they exert different effects on team outcomes. Spatial dispersion is most closely related to reductions in spontaneous communications because it decreases the likelihood of face-to-face communication (Allen, 1977, Te'eni, 2001). TD, however, not only minimizes spontaneous communications but also reduces real-time problem solving because it decreases the potential for synchronous interaction (Burke et al., 1999, Dennis et al., 1988).

In a typical OSS project, software is developed by a spatially-dispersed group that manages interdependencies by coordinating its efforts through computer-mediated channels with limited or no face-to-face interaction (Duchenaut, 2005). The already very low level of face-to-face interaction is not likely to diminish as teams become more distant spatially, but coordination difficulties due to lack of real-time interaction may become more salient as the TD of a team continues to grow (Espinosa et al., 2006). Thus, although spatial dispersion and TD may coexist in OSS projects, their respective mechanisms in affecting coordination are different. This study is concerned about understanding the

effect of TD on OSS project outcomes by influencing the mechanism of asynchronous communication. There has been limited research on the influence of TD on the functioning of virtual teams, and particularly, software development teams. Divergent views exist among the researchers who explicitly or implicitly accounted for TD. On the one hand, several studies suggest that TD, which necessitates the usage of asynchronous communication, detracts from team member coordination and degrades communication quality (Warkentin et al., 1997). In asynchronous communication environments, coordinating temporal patterns of group behavior is a significant challenge because the transmission of verbal cues is hindered, feedback is delayed, and interruptions and long pauses in communication can occur (McGrath, 1991). In addition, long-time lapses between communication events, as is often the case in temporally dispersed situations, can result in disjointed and discontinuous discussions (Ocker et al., 1995). As such, research has found that temporally dispersed virtual teams (e.g., global software development teams) face specific problems, such as increased coordination costs (Espinosa and Carmel, 2003), additional barriers to conflict management (Montoya-Weiss et al., 2001), difficulty in assimilating atypical work hours and in meeting deadlines (Labianca et al., 2005), as well as a substantial decrease in the attainability and effectiveness of leadership control over a team (Jarvenpaa et al., 1998).

On the other hand, some research postulates that TD can have positive effects on team outcomes when asynchronous communication is effectively used. First, asynchronous communication, by definition, eliminates time and space constraints on the act of communicating. Second, asynchronous communication allows members to take time to consider more carefully both the received information and the responses that should follow. Third, it can also allow members to consult other resources, internal or external to the team, for improved problem solving (Borges et al., 1999, Rasters et al., 2002). Fourth, IT tools have been available to coordinate tasks that would otherwise be difficult to manage in asynchronous communication. For instance, in the context of software development, source code control systems (SCCS), also called “Versioning Systems,” such as Concurrent Versioning System or the newer SubVersion, are tools specifically designed to allow developers asynchronously contribute to the code base (Mockus et al., 2002) and to facilitate coordination (Grinter, 2000). OSS project teams also coordinate their work using lean communication media, such as mailing lists (Yamauchi et al., 2000).

The mixed views on the effect of temporal dispersion in existing virtual team literature give rise to the need for developing a deeper understanding of how temporally dispersed teams effectively coordinate their work and whether potential moderating effects exist. Addressing this need requires (1) conceptualizing and measuring temporal dispersion as a distinct construct (rather than approximating its influence by looking at spatial dispersion), and (2) understanding how teams manage temporally interdependent activities, (i.e., perform temporal coordination) for projects of varying complexity. To understand the mechanism of temporal coordination, this study reviews coordination theory and discusses temporal coordination in the next subsection.

2.3 Coordination theory

Building on Thompson’s typology of functional dependencies (Thompson, 1967), coordination theory (CT) defines a team as a group of actors who achieve goals by performing interdependent activities that create or require various types of resources (Malone and Crowston, 1994, Malone et al., 1999). There are two distinct types of activities: goal-oriented and coordination-oriented activities.

The primary goal of software development teams is to write useful source code to complete a functioning program, and thus individual tasks, such as coding and bug fixing, are goal-oriented activities. Coordination activities, on the other hand, are performed to address problems that arise from dependencies which constrain how goal-oriented activities can be performed (Malone and Crowston, 1994). Software project team task assignment and scheduling are examples of coordination-oriented activities.

CT posits that three types of dependencies require coordination efforts: task-task, task-resource, and resource-resource dependencies (Crowston, 1991, Crowston, 1997). Tasks refer to activities

(including both goal- and coordination-oriented activities), whereas resources refer to tools and the time and effort of individuals required for task completion (Crowston, 1997). The necessity of task-task coordination arises from the existence of a set of task precedence relationships unique to a given project, much in the same way that business students learn the critical path method for project management. For instance, task-task coordination is required when the output of one task is the input of others (also known as a prerequisite) and when tasks share the same output (Crowston, 1997).

The need for task-resource coordination is derived from the simultaneous use of shared resources by more than one task. As explained by CT, when multiple tasks share a resource, it is either completely shareable or non-shareable but reusable. If a resource is completely shareable and unlimited, there will be no competition for resources and therefore no need for coordination. However, if a resource is limited, or non-shareable but reusable, the only coordination mechanism possible, as suggested by the theory, is scheduling the use of the resource (Crowston, 1997). In the presence of non-shareable, reusable resources, concentrated teams will be more likely to generate idle manpower and will then become less productive than temporally dispersed teams.

The need for resource-resource coordination arises when the availability of a given resource depends on the presence of another resource (e.g., the availability of a software tool for coding may depend on the availability of a computer). This is the case when one resource is produced by a process in which some other resource is needed. Similar to when there is a set of task precedence relationships, resources will have their own set of resource precedence relationships (Crowston, 1997). Overall, CT states that project team success requires these three coordination activities to be effective, and coordination is of particular importance when dealing with complex tasks with tightly coupled interdependencies such as collaborative software development (Kraut and Streeter, 1995).

2.4 Temporal coordination in temporally dispersed teams

Temporal coordination refers to group actions that manage temporally interdependent activities using various resources with an explicit inclusion of time; this underscores the need to account for time as a scarce resource in coordination activities (Massey et al., 2003). Drawing upon CT, earlier research identifies three types of temporal coordination mechanisms that treat time as an important resource to be managed together with tasks and other resources. They include scheduling, synchronization, and allocation of resources (Marks et al., 2001, McGrath, 1991). Previous research suggests that these temporal coordination mechanisms can influence the nature of team interaction and thus team outcomes (Horton and Biolsi, 1993, Montoya-Weiss et al., 2001).

Scheduling refers to the process of deciding how to coordinate between a variety of possible tasks by specifying the order and allotted time for interdependent tasks. It is an instance of task-task coordination, more specifically, one that specifies when the output of a task is the input of the subsequent task. If a team is temporally co-located, with all members amassed in a given working time, there may be a substantial chance that one or more team members with skills specific to a given task will, at some point, be idle, waiting for the completion of a preceding task. The resulting idle time can be minimized if the team members are correspondingly temporally dispersed in such a way that their allocation will be exactly aligned with the task sequencing derived from the task precedence relationships. Tasks have positive durations, and thus the latter case will require some amount of TD among the team. If temporally dispersed teams can be expected to reduce idle time, then it follows that those teams should be more productive than concentrated teams.

Allocation of resources refers to specifying members with available time to be spent on specific tasks, and therefore is an instance of task-resource coordination. Time is a non-sharable and non-reusable resource. When a member is assigned a task at a specific time, that amount of time becomes dedicated to the specific task by the specific individual and, therefore, cannot be reused anymore. Similarly, members are also resources. If a member is locked in a task, he/she is unavailable to other tasks for the assigned period of time. TD allows for an increase in time resources, such that the time commitment by members can be spread across varying hours.

Synchronization refers to alignment of the pace or effort among members. This is an example of resource-resource coordination in the context of temporal coordination: The alignment of team members across tasks (i.e., resource as individuals) requires that these members be available at the specified time (i.e., resource as time). Using a similar argument to the one used for task-task coordination, the consequent temporal sequencing of resource availability indicates that temporally co-located teams are less productive than temporally dispersed teams. Teams can be temporally allocated so that members are ready to use a resource only when the resource is available, that is, once its own resource precedence relationships are met and not before, in the latter instance requiring coordination efforts that are detrimental to productivity.

In the next section, this study draws on the virtual team literature and the coordination theory discussed in the present section to build research models and hypotheses on the performance impact of TD in the context of OSS projects.

3. Research model and hypotheses

In developing the research model, this study focuses on understanding the effect of TD on both the quality and quantity aspects of OSS team performance. OSS performance has been extensively studied in prior literature (Colazo et al., 2005, Crowston et al., 2003, Crowston et al., 2004, Raymond and Trader, 1999, Stewart, 2004, Stewart et al., 2005), suggesting a wide array of performance indicators. From among these indicators, quality of code is used to capture the quality aspect of OSS project performance, whereas development (coding) speed is used to capture the quantity aspect.

Following the previous discussion on the general definition of TD for the particular case of OSS project teams, these teams are temporally dispersed for two main reasons. First, OSS projects are hosted in publicly available, internet-based platforms with open membership; individuals from anywhere in the world may register themselves and start participating without consideration of or restrictions on time zone boundaries. Second, although OSS project members can be physically located in the same time zone, different members may have different preferences or restrictions with regard to the time of a day they participate, hence creating a possible TD even within the same time zone.

However, there are several fundamental characteristics of OSS project teams that make temporal coordination distinct from other types of virtual teams. First, OSS project teams feature open and voluntary yet fluid membership. Although achieving core developer status is not a trivial quest (Qureshi and Fang, 2010), most OSS projects make it extremely easy for individuals to join in and contribute knowledge, to ensure a sufficient supply of developers (Franke and von Hippel, 2003, Hertel et al., 2003, Lerner and Tirole, 2000, Roberts et al., 2006, Shah, 2006, Von Hippel, 2001, Von Krogh et al., 2003, Wu et al., 2007). They use unrestricted internet-based access and the ability to motivate participants (e.g., hit a programmer's "personal itch") (Raymond, 1999) to make participation in OSS projects possible from anywhere and at any time. Thus, a distinct underlying characteristic of OSS project development is that manpower is an important and theoretically infinite resource, but it is not always enforceable or even obtained due to the voluntary nature of OSS project member participation.

Second, unlike most commercial software development teams that are severely constrained by a set of predefined timelines (Massey et al., 2003), OSS project teams have ongoing programming work without strict deadlines that may forcefully regulate developer efforts (Scacchi, 2002). For many OSS projects, there are even no predefined release dates; the software is released when the core developer group thinks that it is time to release the software (Sarma and Van der Hoek, 2004). Although some projects set loosely defined deadlines, they are not always effective because the voluntary nature of participants makes enforcing those deadlines difficult. In fact, that major features on the critical path are not yet ready as planned happens regularly, leading to frequent timeline adjustments (Michlmayr et al., 2007). Thus, compared with traditional software developers, OSS developers usually have a much bigger latitude over when and how long to complete assigned tasks. Thus, comparatively speaking, time may not be a scarce resource, which is not the case in

commercial software development projects.

Given that human resources in OSS development are fostered by volunteerism, and time as a resource is generally unconstrained in OSS development, project coordination becomes challenging. OSS projects use the central design of software architecture as a facilitator of coordination between tasks and a special subscription-based model to manage temporal coordination between tasks and resources (e.g., coding task and developer), and between multiple resources (e.g., developer and time). These types of coordination are primarily managed through the use of internet-based information technologies, including mailing lists (used for technical discussion, bug-fixing, feature requesting, etc), "to do" lists, and SCCS. Each of these technologies provides an important role in coordinating work across the team: Mailing lists provide a historical log of all communications among developers; bug reporting software keeps track of software errors as they are identified and resolved over time; tracker software logs new features requested by users and developers as well as their implementation; details of tasks pending completion ("to do" lists) are also often available to read and act upon; SCCS data provide a complete history of all changes to the software program that occurred since its inception. With these tools in hand, project members have access to a nearly complete history of the development of codebase, enabling and deepening their engagement in the project.

These technologies, together with the subscription-based model, collectively provide OSS project members with well-structured yet highly flexible temporal coordination mechanisms, allowing the advantages of TD to be better embraced. This study now looks at how the three important aspects of temporal coordination summarized earlier (i.e., scheduling, allocation of resources, and synchronization) are managed in OSS projects through these technologies and in the subscription-based model to achieve an advantage.

First, scheduling in OSS projects is accomplished primarily through the use of a "to do" list or applications of a similar nature (Yamauchi et al., 2000), which identify a list of tasks that need to be delivered. The list indicates the priority of tasks so that interested developers understand which tasks should be completed first. However, developers are still free to tackle tasks of low priority if they so choose. As such, developers are essentially informed about scheduling (a sense of temporal sequence) but not entirely constrained by the schedule, and they may choose to participate during any period suitable for them. In this sense, time is usually a resource for OSS project teams rather than a constraint. Thus, not only will TD not cause coordination problems in scheduling for OSS project teams, but it will also increase the total time available to projects, allow developers to take the extra time needed to code appropriately, and therefore improve team performance both in terms of quantity and quality.

Second and with regard to resource allocation, human resources are allocated based primarily on voluntary subscription in OSS project teams (Markus et al., 2000). With traditional software development teams, allocation of resources is a pertinent coordination issue because both developers and time are limited resources (development cost is measured by man-hours). As such, both must be carefully and optimally aligned to meet pre-specified deadlines (Espinosa et al., 2007). However, for OSS project teams in which manpower is not always guaranteed and time is often not a limited supply, what matters more to resource allocation is whether someone voluntarily subscribes to a task and manages to complete it rather than how much time he/she should take to complete it (Yamauchi et al., 2000). On one hand, such loosely knit task-resource coordination has less need for planning and optimizing the developer resources--coordination tasks that may otherwise be difficult to resolve for temporally dispersed teams (Espinosa and Carmel, 2003). On the other hand, the advantages of temporally dispersed teams are retained, such as more available time for individual developers to reflect, experiment, and code (Borges et al., 1999, Rasters et al., 2002). This "individual" quality time is actually appreciated by OSS project teams (Markus et al., 2000). Thus, more temporally dispersed OSS project teams can be more productive in terms of quantity and quality of coding activity.

Third, synchronization (i.e., aligning the pace or effort among OSS developers) is achieved through a review and selection process using the mailing list and SCCS (Yamauchi et al., 2000). OSS

developers in general have latitude in deciding which tasks to work on and when to work on them; thus, synchronization of these voluntary and seemingly disorganized efforts is particularly important for the success of OSS projects. With traditional project teams, team member efforts are aligned either concurrently through synchronous communication (if the team is temporally co-located) or carefully planned beforehand (if they are temporally dispersed) (Espinosa and Carmel, 2003). The purpose of such alignment is to increase cost-effectiveness or resolve conflicting opinions (Montoya-Weiss et al., 2001).

However, the alignment of these OSS project team resources takes different forms. The code completed by voluntary developers will be made open to public and will be subject to review by core developers before it is approved for inclusion in the code base (Lee and Cole, 2003, Markus et al., 2000). Through peer review, decisions on the inclusion of certain code are made based on technical merit. In other words, developers' efforts, manifested as the delivery of source code, are aligned not based on pre-defined plans or simultaneous communication that is time-sensitive but on a selection and retention process not made under extreme time pressure. In effect, this unique synchronization mechanism, together with ample time available for consumption in the ongoing OSS project, enables (but does not compel) the team members to engage in activities that can enhance the quantity and quality of production (Kelly et al., 1990, McGrath and Kelly, 1986). Furthermore, given that this synchronization process is not time sensitive to OSS projects, TD is not likely to have a negative effect on the synchronization of developers' efforts. On the contrary, greater TD will allow project progress to be continually monitored, reviewed, reflected, and revised around the clock, thereby leading to higher quantity-per-unit time (development speed) and quality of code. Taken together, we hypothesize that

> ***H1(a, b):*** *The degree of temporal dispersion is positively associated with (H1a) the speed and (H1b) the quality of coding in OSS project teams*

However, the relationship between the temporal structure of a team and its performance is not unconditional. The effect of TD on a team's outcomes may be more pronounced when the team is engaged in a project task that is highly complex. When tasks are simple, identifying and understanding how parts of the task affect one other is relatively easier. However, when tasks are more complex, individuals as well as teams find it more difficult to understand and resolve task issues due to the marked increase in the relatedness of task-related activities (Wood, 1986).

In the context of software project teams, software complexity is an important issue for code development. Basili and Hutchens (1983) defined software complexity as a measure of the resources expended while interacting with software code. This complexity arises from the organization of program elements within a program, which impacts activities such as coding, debugging, testing, or modifying the software. The difficulty in performing these tasks depends in good part on the structural characteristics of the source code itself (Kearney et al., 1986). Consequently, this study focuses on structural complexity of the software being developed as a prime contextual factor.

Structural complexity captures the characteristics or attributes of the software artifact being developed or modified. A software artifact can be considered greater in its structural complexity if it contains more interrelated modules or logic flow paths (Darcy et al., 2005). When software is more complex, more information needs to be processed during coding. Developers must not only comprehend each of the individual instructions but must also spend additional cognitive effort to understand how different files, modules, and information flow paths are interrelated. Thus, as the interrelationships among various nested parts of the software artifact increase, developers are increasingly troubled with the challenge of figuring out in real time and under their immediately available resources the interdependencies inherent within the software's logic decoded into computer instructions, and interfacing and/or feeding their solutions to developers undertaking successive or preceding tasks. This challenge may be alleviated when asynchronous communication brought about by TD allows for more time and resources that help understand the nuances of software complexity and arrive at a more effective problem solving. Better problem solving can be translated into an increased number of

bugs being found and fixed, and consequently a higher level of coding activity and code quality. Moreover, TD allows flexible work time allocation for developers. That is, TD keeps them from being constrained to a fixed work schedule and allows them to potentially be in tighter agreement with the natural precedence relations present in a given coding endeavor. By adapting work time to the demands of the interrelated files, developers can minimize idle time and achieve a greater fit between worked hours and productivity. This positive effect is naturally more pronounced when the structural complexity of the overall task is more daunting, and TD may be expected to have a more noticeable positive effect on coding speed and quality when tasks are more complex. Thus, we hypothesize the following:

> ***H2(a, b)****: Software complexity moderates the relationship between TD and (H2a) the speed and (H2b) the quality of coding in OSS project teams in such a way that when software complexity is higher, the effect of TD on the speed and the quality of coding is stronger.*

4. Methods

4.1 Research setting

The public availability of data on OSS, including the product itself (source code) and other artifacts such as mailing lists, change logs, and so on, poses a great opportunity for empirical research. OSS project team information is hosted in web-based repositories that have been used repeatedly as sources of archival data for empirical studies. In accordance with the majority of previous OSS empirical studies (e.g. Colazo and Fang, 2009, Fang and Neufeld, 2006, Hahn et al., 2008), the setting of this research consists of OSS projects hosted in Source Forge (SF) (www.sourceforge.net).

4.2 Sampling

The OSS project data were collected in early 2008. Among all OSS projects hosted in SF, the projects analyzed were restricted to those using “C” as their programming language for two main reasons. First, “C’” is the most popular programming language in SF; second, the use of a single programming language is strongly preferred when using code-based metrics (Jones, 1986). Projects were considered written in “C” when at least 90% of their source code files were written exclusively in that language.

In SF, an overwhelming majority of the registered projects do not have any meaningful activity with no source code at all. Moreover, projects with only one developer are not representative of temporally dispersed project teams or even of projects of interest for business. Bigger teams are more likely to be representative of projects involving the development of popular, widely used software. Consequently and following previous empirical research on OSS, projects with six or more core team members were selected (Crowston and Howison, 2003). In accordance with the previous research (Mockus et al., 2002), core team members are defined here as project members who have administrative rights to write source code in the repository.

In the SF repository, there were 276 software projects being developed by six or more core team members and using “C.” These 276 projects constituted the sampling frame. The final sample was reduced to 100 projects (36% of the sampling frame) because only that number had archived full data on all development activities. In spite of the non-probabilistic nature of the sample (see limitations), it contained projects with varied types of applications (Table 1), sizes, and degrees of maintainability (Oman and Hagemeister, 1994) (Table 2). Data were cross-sectional, and the unit of analysis was the OSS project.

Table 1: Some Projects Sampled	
Project name	Type
cobolforgcc	COBOL compiler
enlightenment	window manager for X11
gaim	instant messaging suite
vim	text editor
zephyr	data acquisition for industrial controllers

Table 2: Some Project Descriptives				
	Min.	Max.	Mean	Median
Source Lines of Code	63	88309	1184	2387
Source Files	5	5651	124	211
4-Metric Maintainability Index	8	172	97	12
Team Members	6	13	8.2	8

4.3 Measurement

From the source code files for each project, quality of coding was measured by the number of total expected pre-test software defects (called "B" after the initial letter of "Bugs") (Ottenstein, 1981), standardized per thousand lines of source code (B/KSLOC). The parameter "B" estimates the expected number of defects latent in the source code, and it has been validated (Gremillion, 1984) against actually found quality defects in the software testing stage. Note that a higher defect count corresponds to lower software quality. This parameter was calculated by parsing the source code of each project with an off-the-shelf program commonly used in software development for obtaining static code metrics (Scitools, 2005).

Product development speed was measured in two ways. First, the average number of days between any two consecutive versions gave a measure of the inter-release time, a direct surrogate for development speed. The SCCS log files provided the version numbers of the software as it was being developed. For each project, all dates when a change in the revision number occurred were recorded for all different types of software versions: major versions (e.g. 1.0, 2.0, etc.), minor versions (1.1 to 1.2 for example), and "build" versions (e.g. 2.1.1 to 2.1.2).

Second, as inter-release times might be affected not only by development speed but also by different release policies, the average number of lines of code written per developer per month, a popular and established measure for development speed, was used as a proxy (cf. Jones, 1986).

Software structural complexity was measured using the project's average McCabe's cyclomatic complexity factor (McCabe, 1976), a widely used metric for assessing the intricacy and understandability of software projects. The projects' source code files were downloaded and parsed with custom-written software analyzer programs to obtain code complexity. The metric was confirmed using two different off-the-shelf code analyzers (Scitools, 2005, Testwell Oy, 2005). Complexity is the moderating variable in this study, and thus the range of complexity in the sample was inspected using the guidelines given by Marciniak (1994). This analysis showed that the programs ranged from simple to highly complex (Table 3).

Table 3: Complexity of Programs

Cyclomatic Complexity Range	Category	%
1-10	Simple	61
11-20	Moderately Complex	23
21 – 50	Complex	14
> 50	Highly Complex to Intractable	2

In extant research, only few attempts have been made to measure TD. Knoll (Knoll, 2000) measured the mean and standard deviation of actual working hours of team members around the Greenwich Meridian Time (GMT). O'Leary et al. (2007) developed an index based on time zone differences among team members. McDonough et al. (2001) measured TD in three ordinal categories: co-located, virtual, and global. This study did not build on the ordinal measure by McDonough et al. (2001) because (1) it compounds the mixed perception of TD and spatial dispersion, and (2) it does not use adequately the information content and the fine granularity of data, where working times are extracted to the exact second when the activity is logged.

Consistent with the working definition of TD used in this paper, a measure was adapted for it based on the variation of actual work hours rather than on time zone differences, as suggested by O'Leary et al. (2007). This decision was made based on two conditions recognized in the literature: the nature of the sample and the source of the data (O'Leary and Cummings, 2007). First, as mentioned earlier, the subjects of the sample, that is, the OSS developers, do not always adhere to normal office hours (Dempsey, 2002). While some may contribute during office time, others may consider OSS projects as a hobby and work at their leisure, which could be at any time of day. Therefore, actual work hours will more accurately capture temporal patterns of OSS developers than time zones. Second, data included the complete details of the OSS developers' specific time for code submission and/or mailing-list activities, thus capturing OSS developers' actual work hours, while location data are far from complete in OSS repositories, making it extremely difficult to obtain complete and accurate data on time zones. Given "the use of any measure involves tradeoffs among precision, ease of calculation, and availability of data" (O'Leary and Cummings, 2007)(p.445), actual-work-hour-based measure was believed to be the best available option for the purpose of the study.

Consequently, TD was measured using the variance in the team members' starting time, where time is expressed in a location-independent time unit: UTC. For every day in a given time window immediately preceding the measurement of the other variables, the time when each developer submitted his/her first contribution was recorded. The mentioned time window was set as a month, but results do not change in significance if the time window is set at a quarter or at a semester.[1]

Developer activity time was observed from two different sources: time stamps in the SCCS log files and stamps recorded in the developers' e-mail lists. SCCS log information was already in UTC regardless of where the code changes came from. E-mail time stamps were not in UTC, but they were transformed into UTC by noting the time zone recorded in the e-mail exchange log (e.g., 10:45 PM UTC+5 was recoded into 5:45 PM UTC). SCCS logs and e-mail logs were then parsed for submission times with custom-made scripts written in Practical Extraction and Report Language (PERL).

With regard to the control variables, the sampling design naturally controlled for programming

[1] Further checks were carried out: TD based on the variance in team members' last (instead of first) contribution of the day was measured, and similar results were obtained. Additionally, as suggested by one of the reviewers, instead of controlling for the team size, we ran an additional model measuring TD by the number of worked hours/ (total worked hours x number of developers) with the intention of standardizing the metric. The results were again agreeable (available on request).

language to eliminate inconsistencies in the metrics based on source code, such as the number of defects or the complexity. Also, the TD measure naturally excludes the level of face-to-face communication, which is near zero in most OSS projects regardless of the degree of spatial dispersion (Duchenaut, 2005). Other controls included project tenure, team size, source code size, type of software license, and a work time overlap index. Some of these controls were not included in all regressions; for instance, when the dependent variable was Bugs/KSLOC, the size of the source code in KSLOC was not included as a control for obvious model specification reasons.

Project tenure was measured by counting the number of days between the first known date of activity in the source code repository and the date the measurements were taken. Project size was measured in total lines of source code. The number of developers was taken from the SCCS logs. Recent work also suggests that a control should be used for the type of software license used in the project (i.e., whether the license has a "copyleft" clause or not) (Colazo and Fang, 2009, Stewart et al., 2006). Although the effect is still unclear, a dummy variable was used for this purpose, assigning a value of one if the project has a copyleft clause and zero if not.

When measuring TD, introducing an "overlap index" has been suggested to control for the effect on TD of overlapping working times among team members as teams grow larger (O'Leary and Cummings, 2007). This is because in distributed teams, as the team grows in size, work times will tend to overlap and TD will change, and this effect needs to be controlled for. This overlap index was calculated as the average number of developers active in every hour of a 24-hour day for the same time window along which TD was measured (one month).

All variables were log-transformed to increase linearity. TD and task complexity were also standardized, to alleviate colinearity with their interaction cross-product, as it is customary (Aiken and West, 1991).

5. Results

Table 4 shows bivariate correlations and descriptive statistics of the variables used in the regression models. The results of OLS regressions for the three dependent variables (i.e., number of predicted defects per thousand lines of code, average time between consecutive minor releases, and average KSLOC written per developer) are shown in Table 5.

First, H1a, which states that TD is positively associated with development speed, is supported: A significant and negative relationship was found between TD and inter-release time, an indicator of development speed (β=-0.143, $p<0.001$). Although minor releases were used in the presented regression model, the results did not change if either major or "build" releases were used. Furthermore, the result showed that TD was significantly and positively associated with the number of KSLOC written per developer (β=0.076, $p<0.001$), further strengthening the support to H1a.

Second, H1b, which states that TD is positively associated with the quality of coding, was also supported. TD was found to be negatively associated with the number of total expected pre-test software defects per thousand lines of code (β=-0.014,$p<0.05$), indicating that high TD is associated with fewer defects and subsequently with a higher quality of coding.

Third, H2a, which states that software structural complexity moderates the relationship between TD and development speed, was not supported. No significant interaction effects of structural complexity and TD were found on inter-release time or KSLOC per developer.

However, H2b, which states that software structural complexity moderates the relationship between TD and quality of coding, was supported. A significant and negative interaction effect on the number of total expected pre-test defects per KSLOC was found between TD and structural complexity (β=-0.046, $p<0.001$), indicating that the positive effect of TD on quality is greater when the complexity of the product is higher. In other words, temporally dispersing a team has a more beneficial effect on quality when the complexity of the product is high than when it is low.

Table 4: Correlations	1	2	3	4	5	6	7	8	9	10	11
1. B/KSLOC	1	-.046	.330 ***	.070 **	.241 ****	.273 ****	.615 ****	.056 **	.221 ****	.097 ****	-.195 ****
2. Inter-Release Time		1	-.053	.048	.076 **	-.072 **	-.106 ***	-.148 ****	.007	-.340 ****	-.085 **
3. KSLOC / Dev			1	-.044	.448 ****	-.193 ****	.894 ****	-.028	.022	.001	.027
4. License (copyleft = 1)				1	.027	-.062 **	-.071 ***	.012	-.066 **	.040 *	-.060 **
5. Team Tenure					1	-.142 ****	.336 ****	.001	-.021	-.096 ****	.006
6. Team Size						1	.334 ****	.515 ****	-.020	.371 ****	.004
7. Source Code Size (KSLOC)							1	.177 ****	,061* **	.208 ****	.023
8. Overlap Factor								1	.080 ***	.339 ****	.106 ****
9. Complexity									1	-.049 *	-.087 ***
10. TD										1	-.180 ****
11. TD X Complexity											1
Mean	1.661	2.252	1.916	0.810	-0.222	0.423	2.060	0.305	0	0	0
SD	0.504	0.403	0.661	0.389	0.247	0.271	0.773	0.010	1	1	1
	****	p < 0.0001	**	p < 0.05							
	***	p< 0.001	*	p < 0.1							

Table 5: OLS Results

	Defects		Development Speed			
	B/KSLOC		Inter-release time		KSLOC/dev	
Intercept	1.749	****	2.223	****	2.671	****
Control Variables						
Copyleft	0.107	****	0.024		-0.124	***
Tenure	0.035		0.131	**	0.658	****
Team size	0.150	****	0.112	**		
Source size			0.000			
Overlap Index	0.005		-0.032		-0.003	
Independent Variables						
TD	-0.014	**	-0.143	****	0.076	****
Complexity	0.047	****	-0.006	*	0.022	*
Interaction						
TD X Complexity	-0.046	****	-0.012		0.028	
R^2	0.13		0.10		0.16	
N = 100						

**** p < 0.001 ** p < 0.05
*** p < 0.01 * p < 0.1

6. DISCUSSION AND CONCLUSIONS

This study represents an original contribution to the literature on virtual teams in general and OSS in particular by providing several insights into how TD relates to OSS team performance.

First, this study supports the claim that TD has a positive effect on OSS development speed when measured both in terms of inter-release time and the number of lines written per developer.

Second, this study provides general support to the claim that TD is positively associated with the quality of coding. The more temporally dispersed an OSS project team, the fewer the expected defects.

Third, this study provides empirical support to the moderating role of one important software structural characteristic: software structural complexity. Software complexity is found to negatively moderate the relationship between TD and the number of predicted defects per KSLOC.

To understand this interaction effect more clearly, the effect is plotted in Figure 1. The results show that when complexity is higher, higher TD is associated with better quality (i.e., lower defect count). This moderating effect implies that TD may increase code quality when working on complex software. This may be because TD allows developers to take time to focus on the problems at hand through comprehension of the received information and the responses that should follow, as well as allowing members to consult other resources for better problem solving effectiveness (Borges et al., 1999, Rasters et al., 2002). When software complexity is low, temporally dispersing team members may not be of great help in terms of code quality.

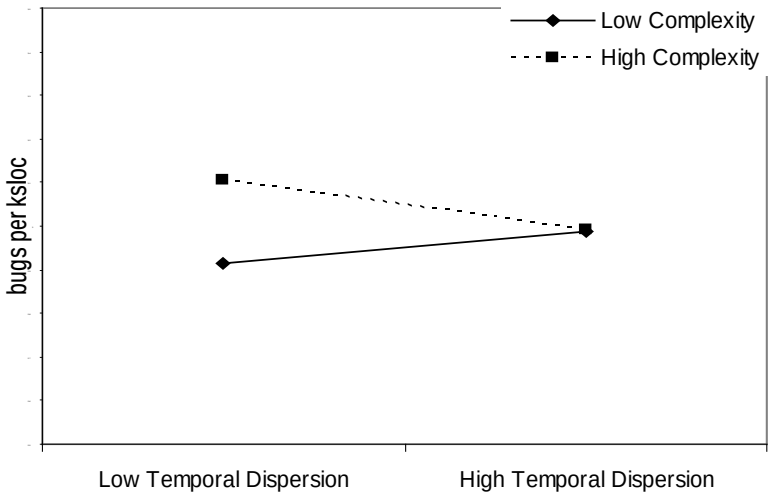

Figure 1. Interaction between TD and software structural complexity

However, from these results, the amount of code written grows with TD but this effect is independent of the code's complexity. A plausible explanation is that the newer code written, while requiring coordination effort, is added at a rate independent of the existing complexity because developers prefer to write more freestanding modules than to dedicate time to understanding the flow of information from older code and building on existing flow paths.

6.1 Research implications

This paper contributes to the literature in several important ways. First, to our knowledge, this is the first empirical study to measure TD and investigate its relationship with project performance in the OSS context, thus adding this crucial yet understudied project structural factor–temporal dispersion– to the mainstream OSS project literature. Specifically, we provided theoretical insights into and empirical support to the important effect of TD on OSS project performance in terms of two different performance dimensions: quality and quantity of coding.

Second, this study contributes to the OSS literature by providing insights into the moderating role of software structural complexity in the relationship between TD and the quality of software development process. Software complexity is an important issue for software development (Kearney et al., 1986) and has serious consequences on the quality of software and on the efficiency of its production process (Banker et al., 1998, Basili and Hutchens, 1983, Kearney et al., 1986, McCabe, 1976, Xia and Lee, 2005). However, existing OSS research that accounts for software complexity is surprisingly rare despite its potentially important consequences to project performance. This research fills this gap by showing that software complexity not only directly influences OSS performance outcomes as expected but also that its effect on quality can be moderated by varying the degree of the team's TD.

Third, this study has meaningful research implications for the virtual team literature. As discussed earlier, the virtual team literature has mixed views on the supposed performance impact of TD (Cramton, 2001, Hinds and Bailey, 2003, Neufeld and Fang 2005, O'Leary and Cummings, 2007,

Warkentin et al., 1997), but without proper empirical evidence (O'Leary and Cummings, 2007). While our TD measure might warrant further refinement, this study breaks new ground with regard to testing the effect of TD. In doing so, this study provides solid, non-perceptual evidence about the relationship between TD and virtual team performance in the context of OSS projects.

This investigation also helps advance the theoretical understanding of the performance impact of TD for virtual team literature. Using OSS as the research context, this study implies that virtual teams, if without time as a strict resource constraint, can better enjoy performance gains by temporally dispersing their team members, particularly when the product being developed is highly complex. In this regard, the findings of this study help provide a more nuanced understanding of the mixed views on the effect of TD on virtual teams by explicitly considering the boundary condition of time and of product complexity.

There are several important practical implications as well. First, the result of this study is encouraging to global virtual team managers who propose that their team's TD results in faster development speed. This paper shows scholarly support for the "Follow the Sun" approach adopted by practitioners. In light of the results of this study, managing the TD of software teams becomes a clear strategic imperative for global project team managers. Second, strategically managing the temporal work patterns of virtual teams in situations where higher-quality work is required for particularly complex programs seems to be especially worthwhile. Conversely, for relatively simpler products, TD will not yield important quality improvements over a temporally uniform team.

7. Limitations and future research

A notable strength of this study is the "hard" nature of the data and metrics. However, limitations with the data should be considered. First, previous research has examined the perils and pitfalls of using SF data in OSS empirical studies, with three major issues: the lack of integrity of the downloaded data, the need to cross-check data with other available sources, and the need to clean the data (Howison and Crowston, 2004). All these problems were carefully addressed and controlled for, mainly through an extensive and costly manual review of all the projects in the sample. For instance, licensing information was crosschecked with licensing files in the SCCS. Samples of the source code were manually inspected to confirm the programming language and line counts. We always used only the main branch of the SCCS, and a small number of projects were built from source and compiled to confirm that they represented somewhat complete software and that no major blocks of code were missing from our analysis. We also double-checked all static metrics using at least two off-the-shelf analyzers.

Second, the sample is non-probabilistic and thus does not represent the universe of OSS projects. However, it is not the intention of this study to generalize findings to small projects or to those that never progressed to the coding phase, which are the majority of the projects hosted on SF and of OSS projects in general (Krishnamurthy, 2002). Instead, setting a lower team size limit means that a sampling bias is introduced and that the bias is in accordance with the kind of software development project that is particularly interesting to the business community (i.e., relatively bigger in team size, with longer time spans and a functioning code base).

Third, some measures may be less appropriate as projects become more sophisticated or use other programming languages. For instance, we averaged complexity at the file level, which does not capture overall architecture. Although in our case we only considered projects written in a procedural language ("C") and cyclomatic complexity is a plausible metric, this measure would need to be reconsidered for object-oriented languages when coupling and cohesiveness become more salient (Chidamber and Kemerer, 1994). Also, in measuring project tenure, since naturally the data may be right-censored one could think that there is a potential bias for overestimation (i.e. at the time of data collection we cannot rule out that the last activity we measured was actually the last activity ever). We eliminated the uncertainty by updating activity data and confirming that all projects in the sample were indeed active beyond our data collection date. It is important to note that our definition of TD is restrictive, in the sense that it does not encompass all possible conceptualizations of dispersion in a

virtual team. For instance, we do not account for the emergence of TD due to member renewal, nor do we account for the dispersion in individual developer effort or efficiency, leaving the task for future studies .

Finally, for the reasons explained in the measurement section, both inter-release times and average SLOC per developer are only two out of potentially many proxies for development speed that can be considered. While our measures arguably do not cover all possible interpretations of "development speed," they offer reasonable support for our results.

Future research can analyze the TD construct within other research frameworks, which will also be useful to assess the external validity of these results. Researchers could also explore the effects of TD on other project outcomes, such as the degree of evolution of the software's functionality as an alternative measure for development speed. Qualitative studies as well as analytical models of TD in teams, considering different moderating effects, can yield rich insights. Cost-benefit analyses of setting up temporally dispersed teams should be of substantive interest to software development managers.

Acknowledgements

This study was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CityU 141809)

References

Aiken, L. S. and S. G. West (1991) *Multiple Regression: Testing and Interpreting Interactions*. Newbury Park: Sage Publications.

Allen, T. J. (1977) *Managing the Flow of Technology*. Cambridge, MA: MIT Press.

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

Banker, R. D., G. B. Davis, and S. A. Slaughter (1998) "Software Developement Practices, Software Complexity and Software Maintenenance Performance: A Field Study," *Management Science* (44) 4, pp. 433-450.

Basili, V. and D. Hutchens (1983) "An Empirical Study of a Syntactic Complexity Family," *IEEE Transactions on Software Engineering* pp. 664-672.

Borges, M. R., J. A. Pino, D. A. Fuller, and A. C. Salgado (1999) "Key Issues in the Design of an Asynchronous System to Support Meeting Preparation," *Decision Support Systems* (27) pp. 269-287.

Burke, K., K. J. Aytes, L. Chidambaran, and J. J. Johnson (1999) "A Study of Partially Distributed Groups: The Impact of Media, Location and Time on Perceptions and Performance," *Small Group Research* (30) 4, pp. 453-490.

Chidamber, S. R. and C. F. Kemerer (1994) "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering* (20) pp. 476-493.

Colazo, J. A. (2010) "Collaboration Structure and Performance in New Software Development: Findings from the Study of Open Source Projects," *International Journal of Innovation Management* (In press).

Colazo, J. A. and Y. Fang (2009) "Impact of License Choice of Open Source Software Development Activity," *Journal of the American Society for Information Science and Technology* (60) 5, pp. 997-1011.

Colazo, J. A., Y. Fang, and D. Neufeld. (2005) Development Success in Open Source Software Projects: Exploring the Impact of Copylefted Licenses. *Eleventh Americas Conference on Information Systems*, Omaha, NE, USA, 2005.

Cramton, C. D. (2001) "The Mutual Knowledge Problem and Its Consequences for Dispersed Collaboration," *Organization Science* (12) 3, pp. 346.

Crowston, K. (1991) *Towards a Coordination Cookbook: Recipes for Multi-Agent Action*, MIT Sloan School of Management.

Crowston, K. (1997) "A Coordination Theory Approach to Organizational Process Design," *Organization Science* (8) 2, pp. 145-175.

Crowston, K. and J. Howison. (2003) The Social Structure of Free and Open Source Software Development. *24th International Conference on Information Systems*, Seattle, WA. Individuals, Teams and Virtual Communities.

Crowston, K. and B. Scozzi (2002) "Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development," *IEE Proceedings - Software* (149) 1, pp. 3-17.

Crowston, K., H. Annabi, and J. Howison. (2003) Defining Open Source Software Success. *24th International Conference on Information Systems*, Seattle, WA

Crowston, K., H. Annabi, J. Howison, and C. Masango. (2004) Towards a Portfolio of Floss Success Measures. Collaboration, Conflict and Control. *Proceedings of the 4th Workshop on Open Source Software Engineering*, Edinburgh, U.K., pp. 29-33.

Darcy, D. P., C. F. Kemerer, S. A. Slaughter, and J. E. Tomayko (2005) "The Structural Complexity of Software: An Experimental Test," *IEEE Transactions on Software Engineering* (31) pp. 11.

Dempsey, B. J. (2002) "Who Is an Open Source Software Developer?," *Communications of the ACM* (45) 2, pp. 67-72.

Dennis, A. R., J. F. George, L. M. Jessup, J. F. Nunamaker Jr. and D.R. Vogel (1988) "Information Technology to Support Electronic Meetings," *MIS Quarterly* (12) 4, pp. 591-626.

Duchenaut, N. (2005) "Socialization in an Open Source Software Community: A Socio-Technical Analysis," *Computer Supported Cooperative Work* (14) 4, pp. 323-368.

Espinosa, J. (2003) "Team Boundary Issues across Multiple Global Firms," *Journal of Management*

Information Systems (19) 4, pp. 157-190.
Espinosa, J. A. and E. Carmel (2003) "The Impact of Time Separation on Coordination in Global Software Teams: A Conceptual Foundation," *Software Process Improvement and Practice* (8) pp. 249-266.
Espinosa, J., W. DeLone, and G. Lee (2006) "Global Boundaries, Task Processes and Is Project Success: A Field Study," *Information Technology & People* (19) 4, pp. 345-370.
Espinosa, J. A., S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb (2007) "Team Knowledge and Coordination in Geographically Distributed Software Development," *Journal of Management Information Systems* (24) 1, pp. 135-169.
Fang, Y., and D.J.Neufeld, (2006), "Should I stay or should I go? Worker commitment to virtual organizations", proceedings, *The Hawaiian International Conference on System Sciences*, Hawaii, U.S.A., January 4-6
Fang, Y. and D. Neufeld (2009) "Understanding Sustained Participation in Open Source Software Projects," *Journal of Management Information Systems* (50) 4, pp. 1-50.
Fitzgerald, B. (2006) "The Transformation of Open Source Software," *MIS Quarterly* (30) 3, pp. 587-598.
Franke, N. and E. von Hippel (2003) "Satisfying Heterogeneous User Needs Via Innovation Toolkits: The Case of Apache Security Software," *Research Policy* (32) 7, pp. 1199-1215.
Gallivan, M. J. (2001) "Striking a Balance between Trust and Control in a Virtual Organization: A Content Analysis of Open Source Software Case Studies," *Information Systems Journal* (11) 4, pp. 277-304.
Gartner. (2006) *Databases in an Open-Source World*.
Gonzalez Barahona, J. M., P. de las Heras Quiros, and T. Bollinger (1999) "A Brief History of Free Software and Open Source," *IEEE Software* (16) 1, pp. 32-34.
Gremillion, L. L. (1984) "Determinants of Program Repair Maintenance Requirements," *Communications of the ACM* (27) 8, pp. 826-832.
Grewal, R., G. L. Lilien, and G. Mallapragada (2006) "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management Science* (52) 7, pp. 1043.
Griffith, T. L., J. E. Sawyer, and M. A. Neale (2003) "Virtualness and Knowledge in Teams: Managing the Love Triangle of Organizations, Individuals and Information Technology " *MIS Quarterly* (27) 2, pp. 265-287.
Grinter, R. E. (2000) "Workflow Systems: Occasions for Success and Failure," *Computer Supported Cooperative Work* (9) pp. 189-214.
Hahn, J., J. Y. Moon, and C. Zhang (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.
Hammon, J. S., M. Gerush, and J. Silekis. (2009) Open Source Software Goes Mainstream: Convert Your Cost-Cutting Crisis into an Oss Opportunity. *Forrester Research*.
Hermann, U. (2005) "Unmaintained Free Software," http://www.unmaintained-free-software.org/index.php (February 20th 2008, 2005).
Hertel, G., S. Niedner, and S. Herrmann (2003) "Motivation of Software Developers in Open Source Projects: An internet-Based Survey of Contributors to the Linux Kernel," *Research Policy* (32) pp. 1159-1177.
Hinds, P. J. and D. E. Bailey (2003) "Out of Sight, out of Sync: Understanding Conflict in Distributed Teams," *Organization Science* (14) 6, pp. 615.
Horton, M. and K. Biolsi (1993) "Coordination Challenges in a Computer-Supported Meeting Environment," *Journal of Management Information Systems* (10) pp. 7-24.
Howison, J. and K. Crowston. (2004) The Perils and Pitfalls of Mining SourceForge. *Workshop on Mining Software at the International Conference on Software Engineering*, Edinburgh, Scotland. U.K.
Jarvenpaa, S., L., K. Knoll, and D. E. Leidner (1998) "Is Anybody out There? Antecedents of Trust in Global Virtual Teams," *Journal of Management Information Systems* (14) 4, pp. 29-64.
Jones, C. (1986) *Programming Productivity*. New York: McGraw-Hill.
Kearney, J. K., R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, M. Adler. (1986) "Software Complexity Measurement," *Communications of the ACM* (29) 11, pp. 1044-1050.

Ke, W., and P. Zhang (2009) "Motivations in Open Source Software Communities: The Mediating Role of Effort Intensity and Goal Commitment," *International Journal of Electronic Commerce*, (13) 4, pp. 39-66

Ke, W., and P. Zhang (2010) "The Effects of Extrinsic Motivations and Satisfaction in Open Source Software Development," *Journal of the Association for Information Systems*, in press.

Kelly, J. R., G. C. Futoran, and J. McGrath (1990) "Capacity and Capability: Seven Studies of Entrainment of Task Performance Rates," *Small Group Research* (21) pp. 283-314.

Kirkman, B. L. and J. E. Mathieu (2005) "The Dimensions and Antecedents of Team Virtuality," *Journal of Management* (31) 5, pp. 700-718.

Knoll, K. (2000) "*Communication and Cohesiveness in Virtual Teams*," Doctoral Dissertation, Austin, TX.

Kraut, R. E. and L. A. Streeter (1995) "Coordination in Software Development," *Communications of the ACM* (38) 3, pp. 69-81.

Krishnamurthy, S. (2002) Cave or Community? An Empirical Examination of 100 Mature Open Source Projects, in *First Monday*, vol. 7. Bothell, WA.

Labianca, G., H. Moon, and I. Watt (2005) "When Is an Hour Not 60 Minutes? Deadlines, Temporal Schemata, and Individual and Task Group Performance," *Academy of Management Journal* (48) 4, pp. 677-694.

Lakhani, K. and B. Wolf (2005) Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects., in J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (Eds.) *Perspectives on Free and Open Source Software*, Boston, MA: MIT Press.

Lee, G. K. and R. E. Cole (2003) "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development," *Organization Science* (14) 6, pp. 633-649.

Lerner, J. and J. Tirole. (2000) The Simple Economics of Open Source. *National Bureau of Economic Research Working Paper*.

Majchrzak, A., R. E. Rice, A. Malhotra, N. King et al. (2000) "Technology Adaptation: The Case of a Computer-Supported Inter-Organizational Virtual Team," *MIS Quarterly* (24) 4, pp. 569.

Malone, T. W. and K. Crowston (1994) "The Interdisciplinary Study of Coordination," *ACM Computing Surveys* (26) 1, pp. 87-119.

Malone, T. W., K. Crowston, J. Lee, B. Pentland et al. (1999) "Tools for Inventing Organizations: Toward a Handbook of Organizational Processes," *Management Science* (45) 3, pp. 425-443.

Manzevski, M. L. and K. M. Chudoba (2000) "Bridging Space over Time: Global Virtual Team Dynamics and Effectiveness," *Organization Science* (11) 5, pp. 473.

Marciniak, J. J. (ed.) (1994) *Encyclopedia of Software Engineering*, New York, NY: John Wiley & Sons.

Marks, M. A., J. E. Mathieu, and S. J. Zaccaro (2001) "A Temporally Based Framework and Taxonomy of Team Process," *Academy of Management Review* (26) 3, pp. 473-492.

Markus, M. L., B. Manville, and C. E. Agres (2000) "What Makes a Virtual Organization Work?," *MIT Sloan Management Review* (42) 1, pp. 13-26.

Martins, L., L. Gilson, and M. Maynard (2004) "Virtual Teams: What Do We Know and Where Do We Go from Here?," *Journal of Management* (30) 6, pp. 805-835.

Massey, A. P., M. M. Montoya-Weiss, and Y.-T. Hung (2003) "Because Time Matters: Temporal Coordination in Global Virtual Project Teams," *Journal of Management Information Systems* (19) 4, pp. 129.

McCabe, T. (1976) "A Software Complexity Measure," *IEEE Transactions on Software Engineering* (SE-2) 4, pp. 308-320.

McDonough, E. F., K. Kahn, and G. Barczak (2001) "An Investigation of the Use of Global, Virtual and Colocated New Product Development Teams," *Journal of Product Innovation Management* (18) pp. 110-120.

McGrath, J. (1991) "Time, Interaction and Performance (Tip): A Thoery of Groups," *Small Group Research* (22) pp. 147-174.

McGrath, J. E. and J. R. Kelly (1986) *Time and Human Interaction: Toward a Social Psychology of Time*. New York: Guilford Press.

Michlmayr, M., F. Hunt, and D. Probert (2007) Release Management in Free Software Projects: Practices and Problems, in J. Feller, B. Fitzgerald, W. Scacchi, and A. Silitti (Eds.) *Open*

Source Development, Adoption and Innovation, Boston, MA: Springer, pp. 234-295.
Mockus, A., R. T. Fielding, and J. Herbsleb (2002) "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology* (11) 3, pp. 309-346.
Montoya-Weiss, M. M., A. P. Massey, and M. Song (2001) "Getting It Together: Temporal Coordination and Conflict Management in Global Virtual Teams," *Academy of Management Journal* (44) 6, pp. 1251.
Netcraft (2009) "2009 Web Server Survey," Netcraft, http://news.netcraft.com/archives/web_server_survey.html (September 28).
Neufeld, D.J.and Y. Fang (2005), "Individual, Social and Situational Determinants of Telecommuter Productivity", *Information & Management*, (42), 7, pp. 1037-1049
Neufeld, D.J., Wan, Z. and Y. Fang (2010) "The Effects of Remote Leadership and Communication on Leader Performance", *Group Decision and Negotiation*, 19(3), pp. 227-246
O'Leary, M. B. and J. N. Cummings (2007) "The Spatial, Temporal, and Configurational Characteristics of Geographic Dispersion in Teams," *MIS Quarterly* (31) 3, pp. 433-452.
Ocker, R., S. R. Hiltz, M. Turoff, and J. Fjermestad (1995) "The Effects of Distributed Group Support and Process Structuring on Software Requirements Development Teams: Results on Creativity and Quality," *Journal of Management Information Systems* (12) 3, pp. 127.
Olivera, F., P. S. Goodman, and S. S. L. Tan (2008) "Contrinution Behaviors in Distributed Environments," *MIS Quarterly* (32) 1, pp. 23-42.
Oman, P. and J. Hagemeister (1994) "Construction and Testing of Polynomials Predicting Software Maintainability," *Journal of Systems and Software* (24) 3, pp. 251-266.
Ortega, F., J. M. Gonzalez Barahona, and G. Robles. (2007) On the Inequality of Contributions to Wikipedia. *41st Hawaii International Conference on Information systems*, Waikoloa, HI.
Ottenstein, L. (1981) "Predicting Numbers of Errors Using Software Science," *ACM SIGMETRICS Performance Evaluation Review* (10) 1, pp. 157-167.
Qureshi, I.and Y. Fang, (2009) "Understanding participation behavior and status attainment of open source software developers – a latent class growth modeling approach", *proceedings of Pacific Asia Conference on Information Systems*, Hyderbad, India, July 10-12
Qureshi, I. and Y. Fang (2011) "Socialization in Open Source Software Projects - a Growth Mixture Modeling Approach," *Organizational Research Methods*, 14(1), In press.
Rasters, G., G. Vissers, and B. Dankbaar (2002) "An inside Look: Rich Communication through Lean Media in a Virtual Research Team," *Small Group Research* (33) pp.718-754.
Raymond, E. S. (1999) *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Cambridge, Massachusetts: O'Reilly & Associates.
Raymond, E. S. and W. C. Trader (1999) "Linux and Open-Source Success," *IEEE Software* (16) 1, pp. 85-89.
Roberts, J. A., I.-H. Hann, and S. A. Slaughter (2006) "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management Science* (52) 7, pp. 984-999.
Robles, G. and J. M. Gonzalez Barahona. (2006) Contributor Turnover in Libre Software Projects. *2nd International Conference on Open Source Software*, Como, Italy.
Sarma, A. and A. Van der Hoek. (2004) A Conflict Detected Earlier Is a Conflict Resolved Easier. *4th Workshop on Open Source Software Engineering*, Edimburgh, Scotland, U.K.
Scacchi, W. (2002) "Understanding Requirements for Developing Open Source Software Systems," *IEE Proceedings - Software* (149) 1, pp. 24-39.
Schadler, T. (2004) Open Source Moves into the Mainstream. Forrester Research, Inc.
Scitools (2005) Understand, 2.0 edition, pp. Static Metrics Analyzer: Scientific Toolworks Inc.
Sen, R. (2007) "A Strategic Analysis of Competition between Open Source and Proprietary Software," *Journal of Management Information Systems* (24) 1, pp. 233-257.
Shah, S. K. (2006) "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science* (52) 7, pp. 1000-1014.
Stewart, K. J. (2004) Oss Success: From Internal Dynamics to External Impact. 4th Workshop on Open Source Software Engineering, International Conference on Software Engineering, Edinburgh, UK.
Stewart, K. J. and T. Ammeter. (2002) An Exploratory Study of Factors Influencing the Level of

Vitality and Popularity of Open Source Projects. *Twenty-Third International Conference on Information Systems, Barcelona, Spain*.

Stewart, K. J., T. Ammeter, and L. M. Maruping. (2005) A Preliminary Analysis of the Influences of Licensing and Organizational Sponsorship on Success in Open Source Projects. *38th Hawaii International Conference on System Sciences*, Hawaii, HI.

Stewart, K. J., A. P. Ammeter, and L. M. Maruping (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.

Te'eni, D. (2001) "Review: A Cognitive-Affective of Organizational Communication of Designing It," *MIS Quarterly* (25) 2, pp. 251-312.

Testwell Oy (2005) Cmt++, 4.2 edition, *Software Static Metrics Analyzer*. Hermia, Finland: Testwell.

Thompson, J. D. (1967) *Organizations in Action*. New York: McGraw-Hill.

Townsend, A. M., S. M. DeMarie, and A. R. Hendrickson (1998) "Virtual Teams: Technology and the Workplace of the Future," *The Academy of Management Executive* (12) 3, pp. 17.

Von Hippel, E. (2001) "Innovation by User Communities: Learning from Open Source Software," *MIT Sloan Management Review*, Summer, pp. 82-86.

Von Krogh, G., S. Spaeth, and K. Lakhani (2003) "Community, Joining and Specialization in Open Source Software Innovation," *Research Policy* (32) pp. 1217-1241.

Warkentin, M., L. Sayeed, and R. Hightower (1997) "Virtual Teams Versus Face-to-Face Teams: An Exploratory Study of a Web-Based Conference System," *Decision Sciences* (28) 4, pp. 975-997.

Wood, R. E. (1986) "Task Complexity: Definition of the Construct," *Organizational Behavior and Human Decision Processes* (37) 1, pp. 60-82.

Wu, C.-G., J. H. Gerlach, and C. E. Young (2007) "An Empirical Analysis of Open Source Software Developers' Motivations and Continuance Intentions," *Information & Management* (44) 3, pp. 253-262.

Xia, W. and G. Lee (2005) "Complexity of Information Systems Development Projects: Conceptualization and Measurement Development," *Journal of Management Information Systems* (22) 1, pp. 45-84.

Yamauchi, Y., M. Yokozawa, T. Shinohara, and T. Ishida. (2000) Collaboration with Lean Media: How Open-Source Software Succeeds. *Computer-Supported Collaborative Work*, Philadephia, PA.

About the Authors

JORGE COLAZO is an Assistant Professor in the Department of Business Administration of Trinity University in San Antonio, Texas, USA. He earned a Ph.D. at the Richard Ivey School of Business, University of Western Ontario, Canada. Prior to obtaining his Ph.D. he worked in managerial positions for companies such as Toyota and Unilever, and he is a sought-after consultant in the areas of lean manufacturing and the Toyota Production System (TPS).

His teaching experience includes undergraduate, graduate and executive education in operations management, supply chain management and quantitative decision methods. His research concerns innovation management, the OM-IS interface and lean manufacturing. Dr. Colazo has published in *International Journal of Innovation Management, Journal of the American Society for Information Science and Technology, Professional Safety* and others. He received a doctoral fellowship from the Social Sciences and Humanities Research Council of Canada, several other scholarships, a best paper award in the Americas Conference on Information Systems (AMCIS) 2005 and a nomination for best paper award in the Hawaii International Conference for System Sciences (HICSS) 2010.

YULIN FANG is an Assistant Professor in the Department of Information Systems, City University of Hong Kong. He earned his Ph.D. at Richard Ivey School of Business, University of Western Ontario. His current research is focused on knowledge management, virtual teams, and open source software projects. He has published papers in major journals such as the *Strategic Management Journal*, *Journal of Management Information Systems*, *Journal of Management Studies, Organizational Research Methods*, *Journal of the Association for Information Systems, European Journal of Information Systems*, *Journal of the American Society for Information Science and Technology,* among others. He has won the 2009 Senior Scholars Best IS Publication Award, and was the Samsung Best Paper Award Finalist and the Carolyn Dexter Award Finalist at the 2008 Academy of Management Conference.