European Journal of Information Systems (2009) 18, 355–371
© 2009 Operational Research Society Ltd. All rights reserved 0960-085X/09
www.palgrave-journals.com/ejis/

ORIGINAL ARTICLE

Role of collective ownership and coding standards in coordinating expertise in software project teams

Likoebe M. Maruping, Xiaojun Zhang and Viswanath Venkatesh

Information Systems Department, Sam M. Walton College of Business, University of Arkansas, Fayetteville, AR, U.S.A

Correspondence: Likoebe M. Maruping, Information Systems Department, Sam M. Walton College of Business, University of Arkansas, Fayetteville, AR 72701, U.S.A.
Tel: +1 479 575 6840;
E-mail: lmaruping@walton.uark.edu

Abstract
Software development is a complex undertaking that continues to present software project teams with numerous challenges. Software project teams are adopting extreme programming (XP) practices in order to overcome the challenges of software development in an increasingly dynamic environment. The ability to coordinate software developers' efforts is critical in such conditions. Expertise coordination has been identified as an important emergent process through which software project teams manage non-routine challenges in software development. However, the extent to which XP enables software project teams to coordinate expertise is unknown. Drawing on the agile development and expertise coordination literatures, we examine the role of collective ownership and coding standards as processes and practices that govern coordination in software project teams. We examine the relationship between collective ownership, coding standards, expertise coordination, and software project technical quality in a field study of 56 software project teams comprising 509 software developers. We found that collective ownership and coding standards play a role in improving software project technical quality. We also found that collective ownership and coding standards moderated the relationship between expertise coordination and software project technical quality, with collective ownership attenuating the relationship and coding standards strengthening the relationship. Theoretical and practical implications of the findings are discussed.

European Journal of Information Systems (2009) **18,** 355–371.
doi:10.1057/ejis.2009.24; published online 11 August 2009

Keywords: agile development; agile methodology; XP; eXtreme programming; software development teams; software project performance; collective ownership; coding standards

Introduction

Software project teams continue to face tremendous challenges because software development is a complex process that requires intensive coordination of expertise, resources, and work effort. Different software components need to be integrated, production schedules need to be synchronized, and dependencies among tasks and people need to be effectively managed (Espinosa *et al.*, 2007a). Software project teams must manage these coordination challenges as they construct applications to address customer needs. Additional challenges arise when the very nature of 'what' the software is supposed to do continues to change during the software development process (Lee & Xia, 2005). Academics and practitioners alike have highlighted the fact that changing user

Received: 4 May 2008
Revised: 7 October 2008
2nd Revision: 25 June 2009
Accepted: 25 June 2009

requirements continue to be a major challenge in software development (Iansiti & MacCormack, 1997; Lee & Xia, 2005; Maruping *et al.*, 2009). The occurrence of such changes is difficult to anticipate and presents significant challenges to software development teams (Banker & Slaughter, 2000). An inability to develop software under such environmental conditions contributes to an estimated $17 billion in project cost overruns in the U.S. alone (Standish Group, 2003). In order to address the challenges of software development, effective coordination of expertise is required (Curtis *et al.*, 1988; Walz *et al.*, 1993; Akgun *et al.*, 2006).

The importance of expertise coordination in software development is well-recognized in the literature (e.g., Akgun *et al.*, 2006). Complex undertakings, such as software development, require non-routine solutions to various problems that arise, given the idiosyncratic organizational conditions and requirements, thus creating the need for expertise (Argote, 1982; Fry & Slocum, 1984; Boh *et al.*, 2007). Software development teams are typically assembled based on project requirements and expertise availability. However, Faraj & Sproull (2000) suggest that the mere presence of expertise on a software development team is insufficient for project success. Rather a coordinative mechanism is required to ensure that expertise is effectively utilized, when needed, during software development projects (Faraj & Sproull, 2000). Effective coordination of expertise has been found to positively influence performance in software development teams (Faraj & Sproull, 2000; Akgun *et al.*, 2006; Ren *et al.*, 2006). While this prior work underscores the importance of expertise coordination for software development, little is known about how development methodologies support such coordinative efforts.

In response to the ever-increasing need for adaptiveness in software development, numerous methodologies have emerged over the years. *Agile methods* are a recent manifestation of efforts to improve adaptiveness in the software development process (Fowler & Highsmith, 2001; Rico, 2008; Conboy, 2009) that is also fairly widely used, albeit to varying extents (Cusumano *et al.*, 2003; Fitzgerald *et al.*, 2006). Although the specific practices that comprise each of these methods differ, the methods themselves adhere to a common set of underlying principles that emphasize simplicity, flexibility, and responsiveness to change in the design and development of software (Fowler & Highsmith, 2001; Highsmith & Cockburn, 2001). Agile methods have received significant attention both in the popular press (e.g., Fowler & Highsmith, 2001; Nerur *et al.*, 2005) and are beginning to be examined in academic research as well (e.g., Baskerville *et al.*, 2002; Boehm & Turner, 2005; Fitzgerald *et al.*, 2006; Pikkarainen *et al.*, 2007). Although agile methods are implicitly assumed to provide greater adaptiveness in software development, to date, evidence of their effectiveness has been largely anecdotal (e.g., Poole & Huisman, 2001; Murru *et al.*, 2003). In a recent systematic review of over 1900 studies of agile software development, only 36 of them were identified as being empirical studies (Dyba & Dingsoyr, 2008). Based on their review of the agile development literature, Dyba & Dingsoyr (2008) and Erickson *et al.* (2005) call for more and better empirical studies of agile software development.

Effective coordination in software project teams requires a process for mobilizing developer expertise to solve emergent problems (Faraj & Sproull, 2000; Espinosa *et al.*, 2007a). In addition, a common understanding of software code is necessary to enable developers to collaborate in solving such problems (Espinosa *et al.*, 2007b). An important, yet hitherto unanswered, question relates to how well agile methods facilitate the effective coordination of expertise in development teams. As a set of methods, agile development denotes specific activities and procedures which, if appropriately executed, enable specific outcomes to be achieved (Iivari *et al.*, 1998). Thus, it is important to understand *whether* and *how* the practices that comprise specific agile methods facilitate effective coordination of expertise. More broadly, the effectiveness of teams is determined by the processes and work practices they employ (Cohen & Bailey, 1997; LePine *et al.*, 2008). We, therefore, chose one team process construct and one work practice construct. Much like team process variables have been of great interest lately (see Ilgen *et al.*, 2005 and Marks *et al.*, 2001), there is a growing interest in how different work practices impact performance (see Perlow *et al.*, 2004).

While there are few different general constructs related to team process (e.g., team monitoring) and work practices (e.g., workload sharing), there have been calls, both in IS and management, to develop theory by giving rich consideration to the context – that is, the situational opportunities or constraints that influence organizational behaviors and functional relationships between variables (see Orlikowski & Iacono, 2001; Johns, 2006). Contextual factors can be considered at different levels of abstraction (e.g., organizational context, project context, team level context, personal context) and these factors could either independently or interdependently affect the theorized relationship. For example, an organizational level context might moderate the theorized relationship independently or it might interact with a team level context to affect the theorized relationship. However, the inclusion of contextual factors at different levels and the theorizing about their interdependencies can obstruct theoretical clarity in a single study. Given our focus on agile methods, we examine contextual factors that are enacted at the level of the team. From this perspective, exposure to these contextual factors ought to be homogeneous within teams and variable across teams (Morgeson & Hofmann, 1999).

The purpose of this research is to understand how the coordination of expertise can be managed using agile development practices. To accomplish this objective, we focus on one specific agile development method – eXtreme Programming (XP). XP is one of the most broadly recognized agile methods and is being increasingly adopted in organizations (Baskerville *et al.*, 2002; Erickson *et al.*, 2005; Pikkarainen *et al.*, 2007). We draw on the agile development literature and identify two specific practices, one related to team process and the other being a work practice related to programming, namely *collective ownership* and *coding standards* respectively from XP. We focus on these two particular XP practices because they represent formalized activities for managing the flow and use of expertise in software development teams. Thus, by their nature these two practices can be either substitutive for, or complementary to, various aspects of expertise coordination. Both of these XP practices are designed with the purpose of facilitating the coordination of developers' work effort – a foundational element for achieving greater team effectiveness (Beck, 2000; Larman, 2003; Salas *et al.*, 2005). Collective ownership and coding standards are part of a broader set of practices that make up the XP methodology. However, not much is known about whether and how these practices contribute to software project team outcomes (Dyba & Dingsoyr, 2008). Our focus on theorizing the effects of two specific agile practices is not unprecedented in the literature. For instance, a significant proportion of empirical studies of XP have focused exclusively on the pair programming practice (e.g., Williams & Kessler, 2000, 2001; Erdogmus & Williams, 2003; Balijepally *et al.*, 2009). Recent reviews have called for a much more practice-focused approach to understanding agile methods – noting some of the theoretical and practical limitations of adopting a macro-perspective to examining XP (Erickson *et al.*, 2005; Dyba & Dingsoyr, 2008). Thus, we draw on the agile software development literature to shed light on the specific practices that facilitate expertise coordination in software development.

This research is expected to contribute to the literature in a few important ways. First, we develop a theoretical model to understand how expertise coordination and agile practices interact to affect performance in software development projects. Our integrated view of expertise coordination and agile development methods contributes to the expertise coordination literature by identifying two important factors (i.e., collective ownership and coding standards) that strengthen the efficacy of expertise coordination capabilities. Second, we contribute to the software development literature by providing a theoretical rationale for why and when the practices underlying agile methods are useful for enhancing software project team performance. We tested our theoretical model in a four-month field study involving 509 developers organized as 56 software development teams. Thus, this study responds to recent calls for more research on the benefits of software development methodologies (Iivari & Huisman, 2007). Finally, we develop measurement scales for collective ownership and coding standards in order to test their effects in software development projects.

Theoretical background

Agile methods

The agile revolution has seen the emergence of several different methodologies that are designed to manage software development in a dynamic environment. Examples of these methods include XP (Beck, 1999; Holmstrom *et al.*, 2006), Test Driven Development (Beck, 2003), Lean Programming (Poppendeick, 2001), Scrum (Rising & Janoff, 2000; Schwaber & Beedle, 2002; Holmstrom *et al.*, 2006), Crystal (Cockburn, 2001), and Feature Driven Design (Coad *et al.*, 1999). Many of these methods emerged during the mid-1990s and some were established as early as the mid- to late-1980s. These various agile methodologies comprise practices that enable software project teams to cope with the demands of an uncertain development environment. However, the principles underlying each of these agile methodologies are similar. These values, which are outlined in the agile manifesto, emphasize: (1) individuals and interactions over processes and tools; (2) working software over comprehensive documentation; (3) customer collaboration over contract negotiation; and (4) responding to change over following a plan (Fowler & Highsmith, 2001). Currently, organizations are engaging in an increasing amount of experimentation with agile methods, as project managers attempt to evaluate the tangible benefits they provide to software development (Murru *et al.*, 2003; Abrahamsson & Koskela, 2004; Fitzgerald *et al.*, 2006). Recent estimates suggest that about 14% of companies in the U.S. and Europe are using agile methods (North American and European Enterprise Software and Services Survey, 2005). As noted earlier, among the agile methods in use in organizations, XP is the most popular and most widely adopted to date (Baskerville *et al.*, 2002; Erickson *et al.*, 2005; Fitzgerald *et al.*, 2006). It is important to reiterate that XP is only one software development method from a range of agile methods that exist.

Extreme programming (XP)

XP is a lightweight methodology that emphasizes speed and simplicity in software development (Beck, 1999, 2000). It traces its origins to the mid-1990s when Kent Beck was developing DaimlerChrysler's payroll system. Having experienced a simpler and more efficient approach to software development, he formalized the XP methodology (Beck, 1999). XP presents 12 specific practices. They are: the planning game, small releases, use of metaphor, simple design, coding standards, collective ownership, 40-hour work weeks, testing, refactoring, pair programming, continuous integration, and on-site customers (Beck, 2000). These practices reflect the principles outlined in the agile manifesto (Turk *et al.*, 2005). The practices are by no means new to the software development community but rather they represent well-established principles that are carried to the *extreme*.

Software development teams that adopt XP prefer to select specific practices or subsets of practices in their software development projects (Murru *et al.*, 2003; Erickson *et al.*, 2005; Fitzgerald *et al.*, 2006; Pikkarainen *et al.*, 2007; Maruping *et al.*, 2009). Indeed, Fitzgerald (2000) finds this to be true of most methodologies; that is, in practice, software development teams rarely deploy all aspects of the selected development methodology. For instance, Fitzgerald *et al.* (2006) found that the software development teams they studied employed seven XP practices and tailored their use with Scrum practices. Similarly, other studies have found that software development teams in the field rarely adopt all of the XP practices (e.g., Karlsson *et al.*, 2000; Grenning, 2001; Poole & Huisman, 2001; Schuh, 2001; Maruping *et al.*, 2009). Thus, the software development teams employing these practices may not be agile development teams in the strict sense of having fully adopted all practices of an agile methodology. There is some debate as to whether such an *à la carte* approach to development allows a software development team to realize the full benefits of a specific methodology (e.g., Jeffries *et al.*, 2000; Fitzgerald *et al.*, 2003). While engaging in this debate is beyond the scope of this paper, it is a given that teams do employ practices in a piecemeal fashion as a way of gaining greater flexibility in the process of developing software and as there is no practical way of ensuring teams adopt and use all practices of a methodology. It thus becomes important to theoretically and empirically examine the effects of specific practices on key outcomes of interest (Erickson *et al.*, 2005; Dyba & Dingsoyr, 2008). Such an understanding will also provide useful guidance on the value of specific practices that can provide benefits or be harmful so that organizations, teams and leaders can make development decisions accordingly. In sum, this underscores the importance of examining the effects of individual agile practices as opposed to the adoption of the entire methodology. We, therefore, focus our theorizing on the effects of two specific practices, one related to team process and the other related to programming, namely collective ownership and coding standards respectively, that constitute the XP methodology, recognizing that these practices are likely to be employed to varying degrees by software development teams in the field.

Collective ownership encourages all team members to take responsibility for all aspects of the software code (Beck, 2000). Common practice advocates having a specific individual in charge of a specific unit of code. All changes and queries that relate to the code must be relayed through the developer who is in charge of it. In contrast, XP's collective ownership practice enables any developer to change any part of the software code at any time (Beck, 1999, 2000). Such changes include adding functionality, fixing coding errors, and improving code via refactoring. This type of collective ownership structure reduces the need for any single developer to track all activity related to various units of code. Once changes have been made and tested, they can be committed to the main code repository. The use of *coding standards* requires that all developers write and maintain software code in a common and consistent format (Beck, 2000). The rules governing the format of software code facilitate effective communication between developers by providing a common base through which to understand various units of code.

Collective ownership and coding standards are important practices for coordinating developers' efforts in software development projects. Collective ownership is a process that governs the role of developers in managing their own work and contributing to the work outputs of their teammates. The coordination of such work is critical in complex tasks, such as software development, that have high levels of reciprocal interdependence (Thompson, 1967). The idea underlying collective ownership dates back as far as the emergence of joint application development – a development process that coordinates the software coding expertise of developers and the business domain expertise of the client (Wood & Silver, 1989; Carmel *et al.*, 1993). Similarly, for decades, standards have existed as a way of coordinating interactions between transacting entities. The use of standards is pervasive, spanning domains as broad as the international organization for standards (ISO), and domains as narrowly focused as the software engineering institute (SEI), and, even more narrowly focused, coding standards. The use of coding standards is a programming practice that is implemented by developers for the benefit of coordinating collective work. Evidence of the efficacy of these XP practices in enhancing software project outcomes is currently quite limited.

Expertise coordination

Expertise coordination traces its origins to work on transactive memory systems. Transactive memory systems are a mechanism for learning, remembering, and communicating knowledge within a social collective (Wegner, 1986; Lewis, 2003). Wegner (1986) introduced the concept of transactive memory systems in a study of couples. He posited that close couples develop a shared system for encoding, storing, and retrieving information. This shared system is larger than the individual memories of each partner in the relationship. Wegner (1986) and Wegner *et al.* (1991) reasoned that each partner in a couple can focus on developing expertise in a specific domain or set of domains determined, perhaps, by role assignments within the couple. As long as each partner is aware of the other's knowledge, both would enjoy access to detailed memories.

The concept of transactive memory has been adapted to the context of team work. Like couples, teams provide a social context within which individuals are able to tap into the knowledge possessed by their teammates. In teams, transactive memory systems enable team members to maintain an index of who possesses specific expertise. Knowledge of who knows what can emerge through a variety of mechanisms (Wegner *et al.*, 1991). Wegner *et al.* (1991) suggest that such an index of where expertise is located can be developed through role assignment, self-disclosure, or direct observation. When teams have a shared system for accessing knowledge, it reduces the cognitive load placed on team members to encode and store a large amount of specialized knowledge by distributing the process across the team as a whole (Wegner, 1986, 1995). Thus, team members who are unable to recall needed information can retrieve that information from their teammates (Moreland & Myaskovsky, 2000).

Faraj & Sproull (2000) define *expertise coordination* as a team-situated process for managing knowledge and skill dependencies. The concept of expertise coordination as a transactive memory system comprises three key elements: knowing the location of expertise, recognizing the need for expertise, and bringing needed expertise to bear (Faraj & Sproull, 2000). Knowing the location of expertise refers to the extent to which members of a team are cognizant of various potential sources of specialized knowledge. As Faraj & Sproull (2000) point out, such knowledge does not necessarily need be located within the team. Expertise can be located outside of the team in the form of documents, knowledge repositories, or domain experts. In addition to knowing where expertise is located, it is important to recognize when and where expertise is needed on a project. Different types of expertise may be necessary at various stages of a project life cycle. Further, in order to capitalize on existing expertise, when there is a recognized need for it, teams need to be able to bring the expertise to bear – that is, use the expertise to resolve task-related problems, in a timely fashion (Faraj & Sproull, 2000). Although Faraj & Sproull's (2000) concept of expertise coordination has some limitations, Lewis (2003) points out that it is still an ideal measure of transactive memory systems in knowledge worker teams because it is task independent and assesses the existence of the cognitive transactive memory system construct at the level of the team. Finally, expertise coordination extends the notion of transactive memory systems by incorporating the consideration of teams' ability to utilize expertise beyond simply knowing its location.

Hypothesis development

Collective ownership

Collective ownership facilitates an organic approach to software coding. This is similar to a traditional programming practice, namely, code inspection, applied to examine programmers' code in detail to find errors (Panko, 1999). Both collective ownership and code inspection bear the main objectives of finding and fixing coding errors by leveraging the expertise of the entire team. With code inspection, software teams commonly use face-to-face meetings to detect and fix the errors, while with collective ownership, developers are allowed to change software code when necessary. Collective ownership ensures that they are exposed to all aspects of the code (Pikkarainen *et al*., 2007), which gives developers a better understanding of the software architecture and the dependencies that exist among various units of code. This ultimately improves software development teams' capability to create and utilize knowledge (Bahli & Zeid, 2005). It also helps improve the efficiency of the software project team in maintaining the quality of the software code. When software project teams employ a collective ownership structure, team members are empowered to take initiative in addressing problems that arise and in making improvements to the software code where necessary. Under such circumstances, developers do not only focus on the software code for which they are responsible, but also monitor their teammates' code. Developers are free to conduct error fixes whenever they find bugs in their teammates' code (Karlsson *et al*., 2000; Fitzgerald *et al*., 2006). Because developers get exposure to other aspects of the software code besides the code for which they are responsible, they are better equipped to make effective enhancements to software code and to fix errors in the code. They are less likely to inadvertently introduce new errors into the software code. In the absence of collective ownership, developers may not feel compelled to monitor their teammates' software code. Errors or software coding inefficiencies (such as functionality duplication) can go unnoticed if such a collective sense of ownership is not in place. With no collective ownership, developers may be highly protective of the code for which they are responsible. Any changes to a specific unit of code need to be coordinated through the developer who is responsible for it (Beck, 2000). This can create bottlenecks in the development process. In sum, other things being equal, we expect software project teams with collective ownership to produce software code that is of higher technical quality (fewer coding errors) than software project teams that do not. Therefore, we hypothesize:

H1a *Collective ownership is positively related to software project technical quality (i.e., it should negatively relate to the number of errors in the software code).*

Faraj & Sproull (2000) argue that in order to coordinate developer effort on complex software programming tasks, it is important to know where the necessary expertise is located. This enables a developer to seek help from the person who is most likely to have problem-relevant expertise. Access to such expertise is important when a developer is facing non-routine challenges in developing software code. When a collective ownership process is in place in a software project team, every team member is responsible for the whole software application. This shared responsibility gives team members a useful mental map of the various components of the software. Collective ownership empowers developers to take a more active role in improving other's software code. Thus, if developers on the team face non-routine challenges, other team members with the requisite expertise are free to contribute to that developer's code (Larman, 2003). Under this type of ownership process, developers do not necessarily need to know the location of expertise. In other words, because collective ownership facilitates a much more organic, free-flowing approach to software development problem-solving, it is less important for the software project team to have the mental indexing system provided by knowing who knows what. In contrast, the absence of a collective ownership process makes it important for team members to have an understanding of the location of expertise. When challenges emerge in software development, developers need to know who to go to for assistance (Faraj & Sproull, 2000). With low collective ownership, other developers are less likely to take the initiative to solve coding problems that fall outside of their set of responsibilities. As a result, software code limitations are likely to persist unless one actively seeks help. In sum, we expect collective ownership to attenuate the relationship between knowing the location of expertise and software project technical quality. Therefore, we hypothesize:

H1b *Collective ownership will moderate the relationship between knowing the location of expertise and software project technical quality (i.e., the number of errors in the software code) such that the relationship will be weaker when collective ownership is high.*

An important part of exploiting the knowledge resources at a team's disposal is recognizing when such resources are actually needed. Teams that are able to recognize the need for expertise are in a position to mobilize the appropriate resources to solve problems (Faraj & Sproull, 2000). When team members fail to recognize the need for expertise, significant time and resources can be wasted in trying to develop a solution. Further, the resulting solution is likely to be of lower quality than one developed using the appropriate expertise. The fluid ownership structure provided by collective ownership ensures that problems in the software code do not go unnoticed and unaddressed. Developers routinely check each other's work and make improvements in the software code where necessary. In contrast, when the level of collective ownership is low, team members are less likely to check or improve each other's work. The ability to recognize when and where expertise is needed within the team becomes increasingly important under these conditions. Software project teams must rely on such capabilities in order to effectively diagnose and marshal necessary resources to address complex software coding challenges (Boh *et al.*, 2007). Otherwise coding deficiencies are likely to go unnoticed, resulting in low quality software. Therefore, we hypothesize:

H1c *Collective ownership will moderate the relationship between recognizing the need for expertise and software project technical quality (i.e., the number of errors in the software code) such that the relationship will be weaker when collective ownership is high.*

At high levels of collective ownership, team members need to take ownership of the whole software application. Even though team members may not be able to take over each other's work, they have some basic knowledge or understanding of other's work. In most cases, team members get to know each other through coordination and collaboration that generally occurs when team members need to take ownership of the whole application. Such coordination and collaboration enhances team members' experience in how to be more effective in working with each other. In other words, when collective ownership is high, team members are likely to develop effective coordination or collaboration processes. In bringing expertise to bear, team members create more opportunities for sharing of knowledge, especially tacit knowledge (Nonaka & Takeuchi, 1995). While bringing expertise to bear positively affects team performance, we argue its effect will be attenuated at higher levels of collective ownership because team members may have already established effective communication and coordination processes, thus placing less demand on creating informal processes to facilitate collaboration and knowledge sharing. In contrast, when the level of collective ownership is low, developers are generally not interested in accessing others' knowledge and they are less likely to communicate and collaborate. In this case, the role of bringing expertise to bear becomes more important in terms of facilitating collaboration and knowledge sharing. In sum, when there is a low level of collective ownership, the effect of bringing expertise to bear on software project technical quality would be strengthened. Therefore, we hypothesize:

H1d *Collective ownership will moderate the relationship between bringing expertise to bear and software project technical quality (i.e., the number of errors in the software code) such that the relationship will be weaker when collective ownership is high.*

Coding standards

Coding standards require developers to follow a clearly defined, mutually agreed coding practice in developing software code (Beck, 2000). An important basic principle underlying any unit of software code is that it communicates its purpose and structure to other developers who may not have coded it (Martin, 2002). Coding standards provide a common base for exchanging and understanding work outputs of individual developers. Adherence to a common software coding standard enables developers to comprehend each other's code. Significant cognitive resources can be expended in trying to comprehend software code that does not adhere to recognized standards within a project (Banker *et al.*, 1998). For example, if a developer uses a special approach to designing objects, other team members who are not familiar with the approach will have difficulty in understanding the code. This can be problematic when a unit of software code needs to be integrated with other software components in a project. The adherence to coding standards makes it easier to integrate various units of code as the purpose and structure of each unit of code is clearly outlined and commonly understood by developers on the project team (Chong, 2005). Espinosa *et al.* (2007b) underscore the importance of such familiarity for software project performance. As developers work together to develop the project, there is less chance of errors being introduced into the software code. Therefore, we hypothesize:

H2a *Adherence to coding standards is positively related to software project technical quality (i.e., it should negatively relate to the number of errors in the software code).*

Coding standards provide an important platform for coordinating expertise in software project teams. When developers adopt a common standard for constructing software code, it becomes easier to identify the right expertise that is required when unique coding challenges arise (Kraut & Streeter, 1995). Specifically, developers become more efficient at locating expertise within the team because they are able to comprehend the code and, consequently, the specific expertise that is needed to solve the problem. Developers are thus able to quickly get the help they need from the correct sources when non-routine problems arise (Curtis *et al.*, 1988). Indeed, Espinosa *et al.* (2007b) noted that familiarity with task elements and familiarity with team members form complementary factors that enhance the performance of software project teams. When developers do not adhere to coding standards, there is no common base from which to understand the purpose and structure of their code and it would become difficult to identify the right expertise to solve emerging problems. Developers may receive erroneous guidance if they solicit help from a non-expert. This can occur when the purpose and structure of a unit of code is ill-understood. Even if the correct expert is identified, he or she may have limited impact in helping to address emerging challenges if they have no good basis for understanding the software code's purpose and structure. In sum, knowing the location of expertise is most beneficial when a software project team has coding standards that are adhered to. Therefore, we hypothesize:

H2b *Adherence to coding standards will moderate the relationship between knowing the location of expertise and software project technical quality (i.e., the number of errors in the software code) such that the relationship will be stronger when coding standards are place.*

At high levels of adherence to coding standards, team members develop an application by following the same practices. In this case, they are more likely to develop similar mental structures for developing software applications (Chong, 2005). As an example, they are likely to use the same style of creating software objects to increase flexibility and reusability, or think carefully about how the code they create would affect others' code. When team members develop similar mental structures for addressing coding problems, they have a better understanding of each other's work practices and solutions. This enables them to engage in perspective taking – a core ingredient for mutual problem-solving (Boland & Tenkasi, 1995). Consequently, team members should collaborate more efficiently and effectively. As both coding standards and recognizing the need for expertise affect team performance in similar ways – that is, make team members collaborate more efficiently and effectively, the effect of either one of them on team performance would be stronger when the other one is absent. That is, when the level of coding standards is low, it is more important for the team to recognize the need for expertise to facilitate collaboration among team members. In contrast, at high levels of coding standards, team members are likely to collaborate well and team's dependence on recognizing the need for expertise would be reduced. Therefore, we hypothesize:

H2c *Coding standards will moderate the relationship between recognizing the need for expertise and software project technical quality (i.e., the number of errors in the software code) such that the relationship will be weaker when coding standards are in place.*

As noted earlier, coding standards enable team members to develop a shared understanding of the structure and purpose of software code. This common understanding facilitates the transfer of tacit knowledge among developers in the team (Espinosa *et al.*, 2001; Reagans & McEvily, 2003). The ability to effectively bring needed expertise to bear on software development challenges is dependent on the transfer of such knowledge through interpersonal exchange (Faraj & Sproull, 2000). Indeed, Faraj & Sproull (2000) note that problem solving in software development occurs through an emergent process of shared expertise. Coding standards form a common base for such sharing of expertise. When coding standards are adhered to, developers are able to more effectively understand and diagnose problems because there is less ambiguity about the structure and purpose of the software code. When the structure and purpose of the software code is clear and jointly understood, developers are able to integrate outputs and solve problems with greater ease. In contrast, when coding standards are not adhered to, interpersonal exchange of information can still occur. However, misunderstandings and confusion are likely to emerge and can potentially lead to software coding errors (Walz *et al.*, 1993). In sum, the process of bringing needed expertise to bear is most beneficial for software project technical quality when coding standards are followed. Therefore, we hypothesize:

H2d *Coding standards will moderate the relationship between bringing needed expertise to bear and software project technical quality (i.e., the number of errors in the software code) such that the relationship will be stronger when coding standards are in place.*

Method

Our research setting involved established organizational software project teams that were involved in developing modules for large enterprise systems over a four-month period. The field study involved three waves of data collection during this period. In the following sections, we briefly discuss details of the scale development and pilot testing of the measures for collective ownership and coding standards. We then discuss the participants, measures, and data collection procedure for the main study.

Scale development

To the best of our knowledge, there are no existing measures for collective ownership and coding standards. We developed new scales for these constructs. It is important to note that both collective ownership and coding standards are conceptualized at the team level. Thus, to be consistent with the level at which these practices are conceptualized, we created scales that employed a referent shift consensus (Klein *et al*., 1994; Chan, 1998; Klein & Kozlowski, 2000). The failure to ensure this consistency could lead to erroneous results (Dansereau *et al*., 1984; Klein *et al*., 1994). As part of a scale development process for a broader set of XP practices, we developed the scales for collective ownership and coding standards by following DeVellis' (2003) guidelines.

Items for collective ownership and coding standards were derived from conceptual definitions, descriptions, and narratives of the agile practices found at practitioner sources, such as AgileAlliance, Beck (2000), and Larman (2003). After the item pools for each construct were created, developers who had experience using XP examined them and provided feedback for addition, deletion, or modification, thus helping us to improve the content validity (Straub, 1989). Then, we performed a card sorting exercise to further refine the scales. We ensured that the scales developed for various XP practices were independent of each other by designing and refining the questionnaires as well as by taking into consideration of the conceptual differences between various XP practices. For instance, a sample item for collective ownership is: 'we all feel a sense of responsibility for the system code'. This reflects the level of role breadth that team members experience and is distinct from, say, the pair programming practice which reflects whether developers physically work in pairs as opposed to individually. It is also distinct from refactoring, which represents development activity aimed at simplifying software code and eliminating redundancies. Following the results of the card sorting exercise, we conducted a pilot test using a field sample of 149 developers, who were involved in different levels of agile projects such that sufficient variability in the measures could be observed. The pilot test showed our scales to have adequate convergent and discriminant validity. The final set of items used to measure collective ownership and coding standards are presented in the Appendix.

Participants

Participants in the study were software developers organized into software project teams in a large software development firm in the U.S. that had launched a number of newly initiated software development projects. We chose teams that were similar in terms of their project time frame. The participating firm encouraged the use of XP practices by its project teams. However, project team leaders had the autonomy to decide which specific XP practices their team used. Consequently, teams varied in the extent to which they enacted various XP practices (Fitzgerald, 2000; Fitzgerald *et al*., 2006). This is fairly consistent with observations from other field studies of XP use in organizations (e.g., Poole & Huisman, 2001; Schuh, 2001; Murru *et al*., 2003; Abrahamsson & Koskela, 2004; Maruping *et al*., 2009). Seventy-three software project teams agreed to participate in our study. Of these 73 teams, 56 provided usable responses at all three waves of measurement, resulting in a response rate of 77%. We checked for non-response bias and found no significant difference in demographics between respondents and non-respondents. Among the total of 509 developers who participated in the study, 142 (28%) of them were women. The average age of participants was 29.4 (SD = 4.9), with an average of 7.22 years of programming experience (SD = 2.54).

Measurement

Expertise coordination: Expertise coordination was measured using an 11-item scale adapted from Faraj & Sproull (2000). The scale captures the extent to which team members knew the location of expertise in their team, recognized the need for expertise, and were able to bring needed expertise to bear. The reliability for this scale was 0.82. The ICC(1) and ICC(2) were 0.14 and 0.73 respectively, suggesting adequate team-level properties.

Collective ownership: We used a five-item scale to measure collective ownership. The scale measured the extent to which developers on the team were free to make changes to any unit of software code. A 7-point Likert scale was used with 'strongly disagree' and 'strongly agree' as anchors. The scale had a reliability of 0.79. The ICC(1) was 0.17 and the ICC(2) was 0.70 for this scale. This suggested that it was appropriate to aggregate individual team member responses to a team-level score.

Thus, we computed a team-level score by averaging responses from members within each team.

Coding standards: A five-item scale was used to measure the use of coding standards. The items captured the extent to which developers within each team adhered to established software coding standards. A 7-point agreement scale was used. The reliability for the scale was 0.77 and the ICC(1) and ICC(2) were 0.15 and 0.71 respectively, reflecting appropriate team-level properties.

Software project technical quality: Software project technical quality can be assessed in a number of ways, including reliability, complexity, ease of use, and defect rates (Bieman, 2002). The technical quality of software projects was assessed using an objective measure – number of coding errors. This is consistent with prior research that uses number of coding errors to evaluate software team performance with respect to software product quality (e.g., Williams & Kessler, 2000; Ilieva *et al*., 2004; Espinosa 2007b). These objective data were obtained from archival records after the projects were completed, but shortly before delivery of the final product to the client.

Control variables: We controlled for programmer experience, team size, and project size because these variables have been found to affect project outcomes in prior research (see Barry *et al*., 2006). Programmer experience is a team-level score calculated by averaging the years of programming experience of team members within each team. The team size of this study ranged from 8 to 11 members. Project size was measured in terms of the number of lines of code. These data were obtained from archival project records.

Procedure

Seventy-three software project teams worked to build software modules for a large client organization. Led by a team leader whose main responsibility was to manage the progress of the project by addressing project-related challenges and evaluating team effectiveness, each team was to build a specific module for an enterprise-wide system. We ascertained that there was no interdependence across the modules and the organization wanted the teams to function autonomously as it was critical to ensure completion of the modules within the deadline. The whole project was expected to be completed in four months, offering us the opportunity to collect data at three points throughout the project, that is, at the beginning, during the middle, and at the end. To keep track of responses over time, each team was given a name and participants provided both their names and the name of their own team in the survey. Also, we created a unique bar code for each questionnaire to track responses over time. We clearly expressed the purpose of this study to the participants and assured them of the confidentiality of their responses.

The first wave of data was collected one week after the launch of the projects, when team members were asked to report information about their programming experience. The second wave of data was obtained five weeks later when the teams were well into their projects. At this stage, team members were asked to report on their use of collective ownership, coding standards, and expertise coordination over the course of the project. The final wave of data was collected 11 weeks later and included project size and number of coding errors obtained from the archival data.

Results

We first examined construct validity by using factor analysis with direct oblimin rotation, before which we examined normality and linearity of the data found no significant violation of these assumptions. All items loaded on the pre-specified constructs, with loadings greater than 0.70 and all cross-loadings less than 0.20, providing evidence for internal consistency and discriminant validity. Table 1 provides the descriptive statistics and correlations among the constructs in the study. With regard to the correlations to the dependent variable, the control variables, all related to size, were positively correlated with number of coding errors. All five independent variables, related to expertise coordination and the two XP practices, were negatively correlated with the number of coding errors, providing preliminary evidence in favor of the importance of expertise coordination and use of the two XP practices to reduce the number of coding errors. We tested our hypotheses using hierarchical moderated regression analysis. To ensure appropriateness of regression analysis, we examined the data with respect to the underlying assumptions of regression (e.g., normality) and found no violations of ordinary least square regression assumptions.

Model 1 included control variables only. This model indicates how much variance in software project technical quality is explained by project size, team size, and average programming experience. In probability theory and statistics, variance of a random variable is a measure of the statistical dispersion, averaging the squared distance of its possible values from the expected values (mean). Model 2 included the main effects of collective ownership, coding standards, and expertise coordination. This model indicates how much variance in the software project technical quality is explained by the main effects beyond what is explained by the control variables. Keeping in mind that our dependent variable is the number of errors: when there is a positive relationship between the predictor and the dependent variable, the number of errors increases as the value of the predictor increases, indicating the negative impact of the predictor on software project technical quality (since more coding errors signify lower software technical quality). In contrast, a negative relationship between the predictor and the dependent variable indicates a positive impact of the predictor on software project technical quality (since fewer coding errors signify higher software technical quality). Therefore, the hypothesized relationships of 1a and 2a will be supported if the

Table 1 Descriptive statistics and correlations

Variables	*Mean*	*SD*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1. Number of coding errors	166	34.5								
2. Project size	566,650	79,678	0.22**							
3. Team size	10	1.33	0.16*	0.17*						
4. Avg. programming experience	7.22	2.54	0.19**	0.20**	0.17*					
5. Expertise location	3.86	1.22	−0.22**	0.14*	0.14*	0.12				
6. Expertise needed	4.01	1.28	−0.20**	0.08	0.12	−0.14*	0.22**			
7. Expertise to bear	4.04	1.21	−0.13*	0.14*	0.20**	0.08	0.25***	0.23**		
8. Collective ownership	4.35	1.38	−0.22**	0.13*	0.14*	−0.15*	0.17**	0.08	0.18**	
9. Coding standards	4.41	1.40	−0.21**	0.08	0.17**	0.19**	0.18**	0.14*	0.16*	0.18*

Notes: $n=56$; *$P<0.05$; ** $P<0.01$; ***$P<0.001$.

Table 2 Results of regression predicting software project technical quality

	DV: Number of coding errors			
Variables	*Model 1*	*Model 2*	*Model 3a*	*Model 3b*
Project size	0.14*	0.13*	0.13*	0.13*
Team size	0.13*	0.04	0.02	0.01
Average programming experience	−0.13*	0.02	0.00	0.02
Expertise location		−0.15*	−0.12*	−0.13*
Expertise needed		−0.14*	−0.03	−0.04
Expertise to bear		−0.08	−0.07	−0.04
Collective ownership		−0.17**	−0.14*	−0.11
Coding standards		−0.13*	−0.10	−0.14*
Collective ownership × expertise location			0.28***	
Collective ownership × expertise needed			0.08	
Collective ownership × expertise to bear			0.04	
Coding standards × expertise location				−0.29***
Coding standards × expertise needed				0.08
Coding standards × expertise to bear				−0.21***
R^2	0.07	0.17	0.25	0.30
ΔR^2		0.10	0.08	0.13

Notes: $n=56$; *$P<0.05$; **$P<0.01$; ***$P<0.001$.

results indicate negative and significant coefficients between the predictor and the dependent variable. Model 3a examines the interaction between collective ownership and expertise coordination and model 3b examines the interaction between coding standards and expertise coordination. Both models examine whether the effect of the predictor on the dependent varies across different levels of the moderator (i.e., collective ownership in model 3a corresponding to hypotheses 1b, 1c, and 1d and coding standards in model 3b corresponding to hypotheses 2b, 2c, and 2d). Finally, following guidelines outlined by Aiken & West (1991), collective ownership, coding standards, and expertise coordination were standardized prior to creating the interaction terms to reduce collinearity between the main effects and interaction terms.

The regression results are presented in Table 2. First, we present the results of the model with only control variables (i.e., model 1). As we can see, project size ($\beta=0.14$, $P<0.05$) and team size ($\beta=0.13$, $P<0.05$) were positively related to the number of coding errors, while average programming experience ($\beta=-0.13$, $P<0.05$) was negatively related to the number of coding errors. All three control variables account for 7% of the variance in software project technical quality. Next, we present the results of the main effects model (i.e., model 2). The main effects model explained 17% ($\Delta R^2=0.10$) of the variance in software project technical quality, among which 10% is attributed to expertise coordination and the two agile practices. Consistent with prior research, expertise location ($\beta=-0.15$, $P<0.05$) and recognizing the need for expertise ($\beta=-0.14$, $P<0.05$) were negatively related to the number of coding errors, indicating a positive relationship between expertise coordination and software project technical quality. Bringing needed expertise to bear was also negatively related to number of coding errors but the relationship was not significant ($\beta=-0.08$, $P=$ns). Hypotheses 1a and 2a predicted the relationships between the two agile practices and software project technical quality. The coefficient for collective ownership was negative and significant ($\beta = -0.17$, $P < 0.01$), indicating a positive relationship between use of the collective ownership practice and project technical quality and providing support for hypothesis 1a. Similarly, the coefficient for adherence to coding standards was negative and significant ($\beta = -0.13$, $P < 0.05$), indicating a positive relationship between adherence to coding standards and technical quality and providing support for hypothesis 2a.

Finally, we present the results of the moderated models (i.e., model 3a and 3b). Hypotheses 1b, 1c, and 1d examined the interactive effects of collective ownership and expertise coordination and the results are shown on model 3a in Table 2. The interaction model of expert ise coordination and collective ownership explains 25% of the variance in software project technical quality ($\Delta R^2 = 0.08$). This means that the inclusion of the interaction terms increases the variance explained by 8%. The results indicate that only the interaction between expertise location and collective ownership is significant ($\beta = 0.28$, $P < 0.001$) in predicting software project technical quality. The interaction between recognizing the need for expertise and collective ownership ($\beta = 0.08$, $P = $ ns), and the interaction between bringing the expertise to bear and collective ownership ($\beta = 0.04$, $P = $ ns) were both non-significant. Thus, hypothesis 1b was supported while hypotheses 1c and 1d were not supported. To better understand the pattern of the moderating effect of collective ownership, we plotted the significant interaction by following Aiken & West's guidelines (1991). As illustrated by the slope in Figure 1, the effect of expertise location is much stronger for teams with a low level of collective ownership than for those with high level of collective ownership. We tested the slope of the line (the dotted line) that represents high collective ownership and found the slope is not significantly different from zero, indicating the effect of expertise location on software project technical quality stays the same when there is a high level of collective ownership. Figure 1 seems to indicate that there is a higher number of bugs produced among developers with high collective ownership than are those with low collective ownership as the awareness of expertise location increases. We tested the mean difference of bugs produced between developers with high collective ownership and those with low collective ownership when the level of expertise location is high, but we found that the means were not statistically significantly different. This suggests that when the level of awareness of expertise location is high there will not be much difference among teams, regardless of whether the level of collective ownership is high or low. Therefore, we conclude that teams with different levels of expertise location will result in different performance only when the level of collective ownership is low. Consistent with hypothesis 1b, this interaction pattern suggests that the influence of knowing the location of expertise on software project technical quality weakens with increasing levels of collective ownership.

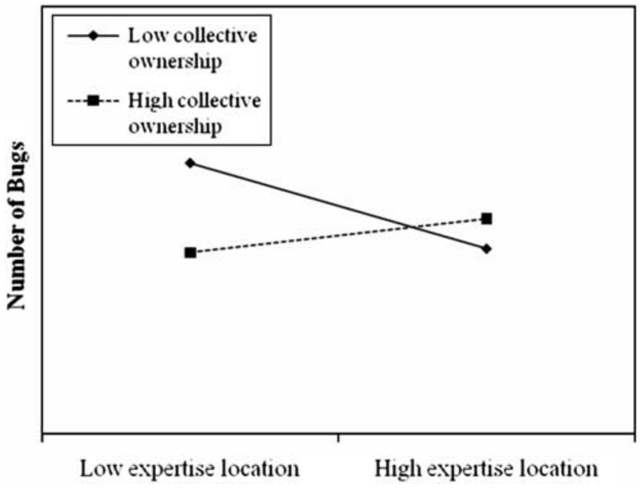
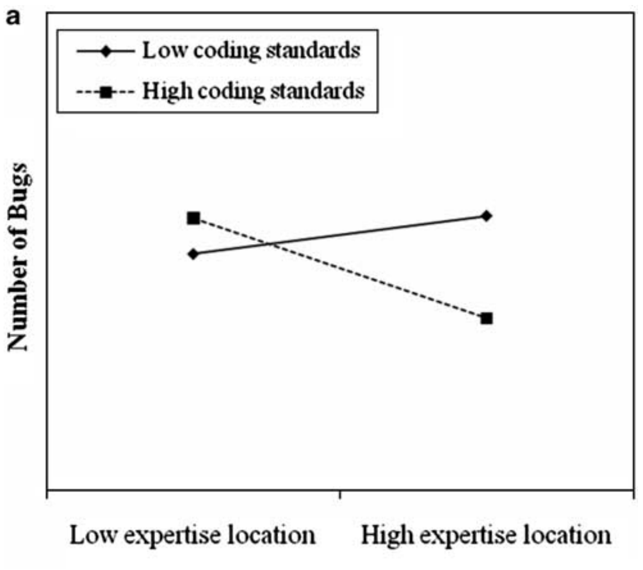


Figure 1 Interaction plot of collective ownership and expertise location.

Hypotheses 2b, 2c, and 2d outlined the moderating effects of adherence to coding standards and the results are shown on model 3b in Table 2. The interaction model of expertise coordination and coding standards explains 30% of the variance in software project technical quality ($\Delta R^2 = 0.13$), suggesting that the inclusion of the interaction terms increases the variance explained by 13%. The results indicate that the interaction between expertise location and adherence to coding standards ($\beta = -0.29$, $P < 0.001$) and the interaction between bringing expertise to bear and adherence to coding standards ($\beta = -0.21$, $P < 0.001$) are significant in predicting software project technical quality. The interaction between recognizing the need for expertise and collective ownership ($\beta = 0.08$, $P = $ ns) was not significant. Thus, hypothesis 2b and 2d were supported but hypothesis 2c was not. In order to understand the form of the moderation, we plotted the significant interactions at one standard deviation above and below the mean for adherence to coding standards. The significant interactions are shown in Figures 2a and 2b. We tested the slope of the line of Figure 2a (the solid line) that represents low coding standards and found the slope is not significantly different from zero. This suggests that the effect of expertise location on software project technical quality is less likely to vary when adherence to coding standards is low. Thus, teams whose members do not adhere to coding standards are likely to see similar quality outcomes, regardless of whether or not team members know where expertise is located. A similar pattern of relationships can be observed in Figure 2b. A test of simple slopes indicated that the relationship between bringing expertise to bear and technical quality is not statistically significant when adherence to coding standards is low (solid line). As the interaction plots indicate, knowing the location of expertise and bringing needed expertise to bear have their strongest effects on software project



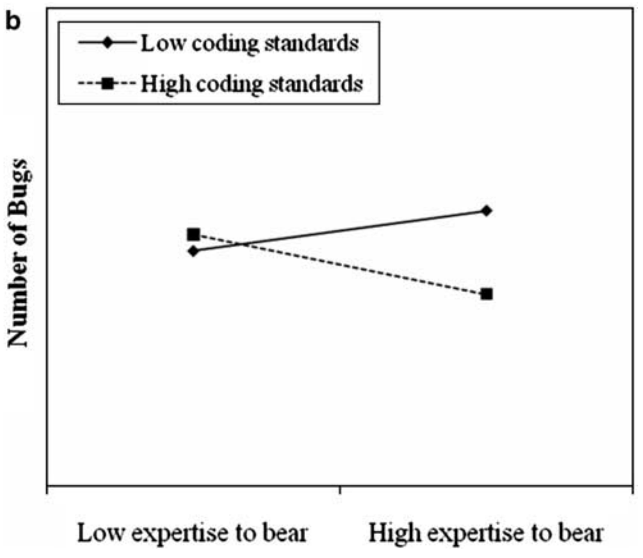


Figure 2 Interaction plot of coding (a) Standards and expertise location; (b) Standards and bringing needed expertise to bear.

technical quality when coding standards are adhered to. This pattern of relationships is consistent with hypotheses 2b and 2d.

Discussion

In this section, we first briefly reiterate the purpose of this paper and summarize the major findings from our study, and then we discuss how this research contributes to theory development and highlight some limitations that need to be addressed in the future. We conclude by discussing the theoretical and practical implications that can be drawn from this research. The objective of this research was to understand how agile development practices facilitate the coordination of expertise in software project teams. We were specifically interested in understanding how expertise could be leveraged in the fast-paced environment characterized by agile development. We reasoned that the XP methodology has practices that are formalized mechanisms for facilitating expertise coordination – namely, collective ownership and coding standards. Drawing on the expertise coordination and agile development literatures, we theorized the processes by which expertise coordination and these two XP practices interact to influence software project technical quality. The results indicated that the main effects model of expertise coordination and the two XP practices explains 17% of the variance in software project technical quality (which we operationalized as the number of coding errors in the software). As predicted, there was a significant interaction between the XP practices and expertise coordination. Specifically, the interaction between collective ownership and expertise coordination explained 25% of the variance in software project technical quality and the interaction between coding standards and expertise coordination explained 30% of the variance in software project technical quality (please see Table 2). To summarize, our theoretical model indicates some factors, such as project characteristics (e.g., project size, team size, and programming experience), XP practices (e.g., collective ownership and coding standards), and coordination among team members, influence software project technical quality independently and interdependently. The remaining 70% of the variance (approximately) may be attributed to other factors (e.g., task complexity) that were not the focus of this study.

This paper contributes to research on agile development and expertise coordination. First, this research contributes to the agile development literature by examining the impact of collective ownership and coding standards on software project technical quality. Previous research has tended to focus more on the adoption of these XP practices (e.g., Murru *et al.*, 2003; Fitzgerald *et al.*, 2006; Pikkarainen *et al.*, 2007). Comparatively, fewer studies have focused on the outcomes of utilizing these XP practices. Thus, by studying the effects of collective ownership and coding standards, we respond to recent calls for research that empirically examines the efficacy of agile practices separately (Erickson *et al.*, 2005; Dyba & Dingsoyr, 2008). The few studies that have examined the outcomes of XP practices examine the more broadly recognized practices, such as pair programming (e.g., Nosek, 1998; Lui & Chan, 2006; Balijepally *et al.*, 2009) and continuous integration (e.g., Karlsson *et al.*, 2000; Ji *et al.*, 2005). We noted that coordination requires a team process for mobilizing expertise in problem-solving and that it was necessary to establish common standards for developer work. We theorized that the use of collective ownership would yield software of a higher technical quality (signified by fewer coding errors) as such a process encourages developers to use their expertise to actively improve on each other's work. We also theorized that adherence to coding standards would lead to better software project technical quality. Previous research has not theorized and empirically tested the relationship between collective ownership, coding standards, and software project technical quality. Together, the theory and empirical findings advance the state of knowledge on agile development.

Second, this research contributes to expertise coordination research. The extant expertise coordination literature stresses the important role of coordination in leveraging existing expertise on a team. Faraj & Sproull (2000) reasoned that such coordination is especially important when teams perform complex and unpredictable tasks. In examining the relationship between expertise coordination and software project performance, Faraj & Sproull (2000) had not considered the role of internal team processes governing member roles and responsibilities. They also did not consider how the efficacy of expertise coordination is affected by standardization of team member work outputs. Hence, we contribute to this corpus of research by incorporating the role of collective ownership and coding standards as critical contextual factors affecting the efficacy of expertise coordination in software project teams. We found that collective ownership and coding standards moderate the relationship between expertise coordination and software project technical quality. Collective ownership substitutes for the role of expertise coordination in improving software project technical quality. Coding standards enhance the effectiveness of expertise coordination in improving software project technical quality. Therefore, this paper extends research on expertise coordination by identifying a key contingency affecting expertise coordination.

Finally, no scales existed to measure the extent to which software project teams use collective ownership and coding standards. We developed a set of scales that we rigorously pilot tested and empirically validated in a field sample of software project teams. These scales possess three important attributes. First, they are focused at the level of the individual practices. This is important because, as we noted earlier, software project teams differ in the extent to which they use various XP practices (Fitzgerald *et al.*, 2003; Pikkarainen *et al.*, 2007). Second, capturing the use of individual agile practices enables researchers to empirically evaluate which specific practices contribute to outcomes of interest (Erickson *et al.*, 2005). Third and finally, the scales are sensitive to levels of analysis issues. Specifically, the scales were developed using referent shift consensus for which the software project team is the referent (Chan, 1998). This ensures that respondents report on the activities of the team as a whole. The scales demonstrated adequate team-level properties (Bliese, 2000).

Strengths and limitations

Before we discuss the implications of our findings, we highlight some strengths and limitations in this study. First, this is a field study on teams that launched their projects within a similar time frame and obtaining such a sample is rare. Such a sample provided us a unique opportunity to examine the processes leveraged by software project teams. Further, even though the number of teams could be improved, the sample size in this study is consistent with prior field research (e.g., Ancona & Caldwell, 1992; Faraj & Sproull, 2000). Second, this study was conducted in the context of a single organization where use of XP was mandatory. While this allowed us to naturally control for inter-organizational differences, it also makes it necessary to replicate these findings in other organizational contexts to increase generalizability. Given the focus of our research on two specific XP practices, we did not include other XP practices in our model. This choice was also driven by power considerations, since we only had a sample size of 56 teams. It will be necessary to examine other XP agile practices that have not been studied. For instance, factors such as testing and refactoring could also contribute to the reduction of coding errors. Also, it would be interesting to find out how motivational factors (e.g., motivational difference between mandatory usage *vs* voluntary usage) and other people factors (e.g., personality) play a role in affecting the results. In addition, future studies should conduct longitudinal analysis and discuss expertise gained during the project. Moreover, future studies should consider a mixed-methods approach that involves collecting qualitative data, so as to enrich our understanding of the agile development phenomenon. Given the number of teams in our sample, it was impractical to try to collect qualitative data from all of the teams involved in the study.

Third, although common method bias could be a concern because we collected data using primarily survey questionnaires, our study design addressed this concern to some extent. As suggested by Podsakoff *et al.* (2003), we measured the independent and dependent variables at different points in time. In addition, we obtained objective data (i.e., number of coding errors) to measure software project technical quality to avoid bias resulting from subjective measures. Moreover, we collected data from multiple members in each team in order to get a more accurate reporting of team activities. Fourth, even though our study demonstrates the important role of expertise in software development, our understanding of expertise could be overly simplistic as we only consider existing expertise. Future research should examine how expertise evolves in terms of ability to learn and obtain new knowledge as older coding expertise might not be useful in new tasks and duties. Similarly, future research should examine the implications of changes in teams' use of XP practices over the duration of a project. Finally, we only included one measure of software project technical quality – number of coding errors. However, technical quality involves many other dimensions that should be included in future research (e.g., robustness, suitability, ease of use). Our focus on number of coding errors was driven by practical considerations within the study context.

Theoretical implications and directions for future research

Fitzgerald (2000) highlighted the prevalence of an *à la carte* approach to employing software development methodologies in organizations. Consistent with this view, the results of our study highlight the value of utilizing collective ownership and coding standards in software development. While prior research on software development methodologies is mostly anecdotal, this paper advances prior research by relating it to research in team process and work practices, the theoretical foundations of which have long been established. The significant influence of these XP practices on software project technical quality indicates the importance of embedding rules and processes that govern how team project work is accomplished. Our identification of these relationships is only an initial step. Collective ownership outlines the process for organizing developer efforts in developing software code. Future research needs to identify the specific developer behaviors that are facilitated by this process. Such behaviors would constitute important mediating mechanisms between collective ownership and software project outcomes. Similarly, the use of coding standards outlines key guidelines that individual developers should follow when writing code. However, it is necessary to understand the collective team actions that are enabled by adherence to coding standards. For instance, the use of coding standards might help improve the development of shared mental models in software project teams (Espinosa *et al.*, 2001, 2007b).

The significant influence of collective ownership and coding standards on software project technical quality stimulates future research to theorize the impact of these software development practices on other software project performance outcomes. Software project performance is a complex outcome encompassing various dimensions including: complexity, size, development cost, customer satisfaction, and on-time completion. In this study, we only focused on one dimension of software project performance – that is, technical quality, indicated by the number of coding errors in the software. It is possible that the efficacy of collective ownership and coding standards varies across different dimensions of team performance because it will be challenging for teams to optimize their performance along all dimensions. For example, it is not uncommon that achieving higher software technical quality on time may result in a product with reduced features. Future research should examine the effect of collective ownership and coding standards on other dimensions of performance. Such research would enable project teams to be more deliberate in aligning their selection of agile practices with key project objectives.

Another major finding in this study is that collective ownership moderates the relationship between expertise coordination and software project technical quality. We argued that collective ownership would substitute for expertise coordination mechanisms because it provides a fluid forum for bringing expertise to bear on problem-solving and software code quality. We found that knowing the location of expertise within the team is important in improving software project technical quality when there is little or no collective ownership. Knowing where expertise is located becomes less important when software project teams engage in collective ownership as developers with the requisite expertise are able to contribute as a natural part of the development process (Larman, 2003). However, it is unclear how well collective ownership enables software project teams to cope with changing environmental conditions that affect software project performance. Recent research has emphasized the importance of expertise coordination under dynamic environmental conditions (Akgun *et al.*, 2006). Future research should examine the efficacy of collective ownership under such environmental conditions.

We also found that the use of coding standards moderates the relationship between expertise coordination and software project technical quality. We argued that the use of coding standards would enhance the effectiveness of expertise coordination by providing a common platform for exchanging and communicating knowledge. We found that knowing the location of expertise was more beneficial for enhancing software project technical quality when coding standards were in place. The use of coding standards also helped enhance the efficacy of bringing needed expertise to bear. Future research is needed to better understand how coding standards can be used to support expertise coordination under dynamic environmental conditions. To summarize, the integration of research in agile development practices and expertise coordination enriches our understanding of how to enhance performance of software development teams. While research in expertise coordination underscores the importance of leveraging the expertise distributed among team members, we contribute to this stream of research by discussing the contextual factors that would play significant roles in affecting the effectiveness of transactive memory systems. Specifically, we highlight the importance of team processes (e.g., collective ownership) and work practices (e.g., coding standards) in affecting the effectiveness of transactive memory systems.

Practical implications

The results of this research highlight the pragmatic benefits of collective ownership and coding standards. One of the challenges associated with expertise coordination is that, because it is a team-situated process, it requires team members to be cognizant of the right resources and the right timing for accessing those resources in response to discrete non-routine events. In contrast, collective ownership routinizes the deployment of expertise in software development. When this process is in place, developers are collectively responsible for all code and those with the requisite expertise to solve problems are expected to contribute their effort to improving the software code where necessary (Beck, 1999, 2000; Larman, 2003). This also helps prevent bottlenecks in the development process when different parts of the software code are owned by specific developers (Larman, 2003). Thus, managers are encouraged to embed this process in software development projects as a way of ensuring high technical quality.

The results of this research also underscore the importance of using coding standards in software development projects. While managers have traditionally been cognizant of the utility of establishing coding standards, this research suggests that the implications of such standards extend to the coordination of expertise. By establishing coding standards, developers in a project team have a common base for making sense of how to identify the necessary expertise to solve non-routine problems and how to bring the needed expertise to bear. Managers can establish new standards that are embraced by the team as a whole or they can utilize programming conventions associated with the specific programming language in which the code is to be written (Larman, 2003). Employing such standards goes a long way in ensuring that fewer coding errors are introduced into the software code as different developers contribute their expertise.

Conclusions

An integrative model of expertise coordination and agile development practices was developed to understand how agile development practices can be leveraged to support the coordination of expertise in software project teams. Collective ownership was argued to substitute for expertise coordination – attenuating its effects on software project technical quality. Coding standards were argued to enhance the effectiveness of expertise coordination in improving software project technical quality. A robust set of scales was developed to measure software project teams' use of collective ownership and coding standards in software development. The integrated model was empirically tested in a field study of software project teams. The findings provide a deeper explication of the complexities of coordinating expertise in software projects.

About the authors

Likoebe M. Maruping is an assistant professor of Information Systems at the Sam M. Walton College of Business at the University of Arkansas. Likoebe's research is primarily focused on the activities through which software development teams improve software project outcomes. His current work in this area focuses on understanding how teams cope with uncertainty in software development projects. He also enjoys conducting research on virtual teams and the implementation of new technologies in organizations. His research has been published or is forthcoming in premier information systems, organizational behavior, and psychology journals including *MIS Quarterly*, *Information Systems Research*, *Organization Science*, *Journal of Applied Psychology*, and *Organizational Behavior and Human Decision Processes*.

Xiaojun Zhang is a Ph.D. student in the Information Systems Department at the University of Arkansas. His research focus is to understand individual job outcomes, such as job performance and job satisfaction. Specifically, his research interests lie at the intersection of knowledge management, technology adoption and use, social networks and culture to understand individual job outcomes.

Viswanath Venkatesh, who completed his Ph.D. at the University of Minnesota in 1997, is a Professor and Billingsley Chair in Information Systems at the Walton College of Business, University of Arkansas. His research focuses on understanding the diffusion of technologies in organizations and society. His work has appeared and is forthcoming in leading information systems, organizational behavior, operations management, marketing and psychology journals. His articles have been cited about 7000 times per Google Scholar and about 3000 times per Web of Science. Some of his papers published in various journals (*Decision Sciences* 1996, *Information Systems Research* 2000, *Management Science* 2000, and *MIS Quarterly* 2003) are among the most cited papers published in the respective journals. He serves and has served on the editorial boards of different journals, including currently serving as a Senior Editor at *Information Systems Research* and as an Associate Editor at *Decision Sciences Journal*, and has served as an Associate Editor at *MIS Quarterly* and *Management Science*.

References

Abrahamsson P and Koskela J (2004). Extreme programming: a survey of empirical data from a controlled case study. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering* (Juristo, F and Shull, F, Eds), pp. 73–82, IEEE Computer Society Press, Redondo Beach, CA.

Aiken LS and West SG (1991) *Multiple Regression: Testing and Interpreting Interactions*. Sage, London.

Akgun AE, Byrne JC, Keskin H and Lynn GS (2006) Transactive memory system in new product development teams. *IEEE Transactions on Engineering Management* **53(1)**, 95–111.

Ancona DG and Caldwell D (1992) Bridging the boundary: external activity and performance in organizational teams. *Administrative Science Quarterly* **37(4)**, 634–665.

Argote L (1982) Input uncertainty and organizational coordination in hospital emergency units. *Administrative Science Quarterly* **27(3)**, 420–434.

Bahli B and Zeid ESA (2005) The role of knowledge creation in adopting extreme programming model: an empirical study. In *ITI 3rd International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society*, pp 75–87, IEEE Computer Press, Cairo, Egypt.

Balijepally V, Mahapatra R, Nerur S and Price KH (2009) Are two heads better than one for software development? The productivity paradox of pair programming. *MIS Quarterly* **33(1)**, 91–188.

Banker RD, Davis GB and Slaughter SA (1998) Software development practices, software complexity, and software maintenance performance: a field study. *Management Science* **44(4)**, 433–450.

Banker RD and Slaughter SA (2000) The moderating effect of structure on volatility and complexity in software enhancement. *Information Systems Research* **11(3)**, 219–240.

Barry EJ, Kemerer CF and Slaughter SA (2006) Environmental volatility, development decisions, and software volatility: a longitudinal analysis. *Management Science* **52(3)**, 448–464.

Baskerville R, Levine L, Pries-Heje J, Ramesh B and Slaughter SA (2002) Balancing quality and agility in Internet speed software development. In *Proceedings of the 23rd International Conference on Information Systems* (Applegate L, Galliers R and Degross JI, Eds), pp 859–864, Association for Information System Press, Barcelona, Spain.

Beck K (1999) Embracing change with extreme programming. *IEEE Computer* **32(10)**, 70–77.

Beck K (2000) *Extreme Programming Explained*. Addison-Wesley, Reading, MA.

Beck K (2003) *Test-Driven Development by Example*. Addison-Wesley, Reading, MA.

Bieman J (2002) Risks to software quality. *Software Quality Journal* **10(1)**, 7–9.

Bliese PD (2000) Within-group agreement, non-independence, and reliability: implications for data aggregation and analysis. In *Multilevel Theory, Research, and Methods in Organizations* (Klein KJ and Kozlowski SWJ Eds), pp 349–381, Jossey-Bass, San Fransisco, CA.

Boehm B and Turner R (2005) Management challenges to implementing agile processes in traditional development organizations. *IEEE Software* **22(5)**, 30–39.

Boh WF, Ren Y, Kiesler S and Busskaeger R (2007) Expertise and collaboration in the geographically dispersed organization. *Organization Science* **18(4)**, 595–612.

Boland RJ and Tenkasi RV (1995) Perspective making and perspective taking in communities of knowing. *Organization Science* **6(4)**, 350–372.

Carmel E, Whitaker R and George J (1993) PD and joint application design: a transatlantic comparison. *Communications of the ACM* **36(4)**, 40–48.

Chan D (1998) Functional relations among constructs in the same content domain at different levels of analysis: a typology of composition models. *Journal of Applied Psychology* **83(2)**, 234–246.

Chong J (2005) Social behaviors on XP and non-XP teams: a comparative study. In *Proceedings of the Agile Development Conference* (ADC'05).

Coad P, De Luca J and Lefebre E (1999) *Java Modeling in Color*. Prentice-Hall, Englewood Cliffs, NJ.

Cockburn A (2001) *Agile Software Development*. Addison-Wesley, Reading, MA.

Cohen S and Bailey D (1997) What makes teams work? Group effectiveness research from the shop floor to the executive suite. *Journal of Management* **23(3)**, 239–290.

Conboy K (2009) Agility from first principles: reconceptualizing the concept of agility in information systems development. *Information Systems Research* **20(3)**, in press.

Curtis B, Krasner H and Iscoe N (1988) A field study of the software design process for large systems. *Communications of the ACM* **31(11)**, 1268–1287.

Cusumano M, Maccormack A, Kemerer C and Crandall B (2003) Software development worldwide: the state of the practice. *Communications of the ACM* **20(6)**, 28–34.

Dansereau F, Alutto JA and Yammarino FJ (1984) *Theory Testing in Organizational Behavior: The Variant Approach*. Prentice-Hall, Englewood Cliffs, NJ.

Devellis RF (2003) *Scale Development: Theory and Applications*. Sage, Thousand Oaks, CA.

Dyba T and Dingsoyr T (2008) Empirical studies of agile software development: a systematic review. *Information & Software Technology* **50(9–10)**, 833–859.

Erdogmus H and Williams L (2003) The economics of software development by pair programmers. *The Engineering Economist* **48(4)**, 283–319.

Erickson J, Lyytinen K and Siau K (2005) Agile modeling, agile software development, and extreme programming: the state of research. *Journal of Database Management* **16(4)**, 88–99.

Espinosa JA, Kraut RE, Lerch J, Slaughter SA, Herbsleb J and Mockus A (2001) Shared mental models and coordination in large-scale, distributed software development. In *Proceedings of the 22nd International Conference on Information Systems*, pp 513–517, Association for Information Systems Press, New Orleans, LA.

Espinosa JA, Slaughter SA, Kraut RE and Herbsleb JD (2007a) Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems* **24(1)**, 135–169.

Espinosa JA, Slaughter SA, Kraut RE and Herbsleb JD (2007b) Familiarity, complexity, and team performance in geographically distributed software development. *Organization Science* **18(4)**, 613–630.

Faraj S and Sproull L (2000) Coordinating expertise in software development teams. *Management Science* **46(12)**, 1554–1568.

Fitzgerald B (2000) Systems development methodologies: the problem of tenses. *Information Technology and People* **13(3)**, 13–22.

Fitzgerald B, Hartnett G and Conboy K (2006) Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems* **15(2)**, 200–213.

Fitzgerald B, Russo N and O'Kane T (2003) Software development method tailoring at Motorola. *Communications of the ACM* **46(4)**, 64–70.

Fowler M and Highsmith J (2001) Agile methodologists agree on something. *Software Development* **9**, 28–32.

Fry LW and Slocum JW (1984) Technology, structure, and work group effectiveness: a test of a contingency model. *Academy of Management Journal* **27(2)**, 221–246.

Grenning J (2001) Launching extreme programming at a process intensive company. *IEEE Software* **16(8)**, 27–33.

Highsmith J and Cockburn A (2001) Agile software development: the business of innovation. *IEEE Computer* **34(9)**, 120–122.

Holmstrom H, Fitzgerald B, Agerfalk PJ and Conchuir EO (2006) Agile practices reduce distance in global software development. *Information Systems Management* **23(3)**, 7–18.

Iansiti M and Maccormack A (1997) Developing products on Internet time. *Harvard Business Review* **75(5)**, 108–117.

Iivari J, Hirschheim R and Klein HK (1998) A paradigmatic analysis contrasting information systems development approaches and methodologies. *Information Systems Research* **9(2)**, 164–193.

Iivari J and Huisman M (2007) The relationship between organizational culture and the deployment of systems development methodologies. *MIS Quarterly* **31(1)**, 35–58.

Ilgen DR, Hollenbeck JR, Johnson M and Jundt D. (2005) Team in organizations: from input-process-output models to IMOI models. *Annual Review of Psychology* **56**, 517–543.

Ilieva S, Ivanov P and Stefanova E (2004) Analyses of an agile methodology implementation. In *Proceedings of the 30th Euromicro Conference*, pp 326–333, IEEE Computer Society Press, Rennes, France.

Jeffries R, Anderson A and Hendrickson C (2000) *Extreme Programming Installed*. Addison-Wesley, Boston, MA.

Ji Y, Mookerjee VS and Sethi SP (2005) Optimal software development: a control theoretic approach. *Information Systems Research* **16(3)**, 292–306.

Johns G (2006) The essential impact of context on organizational behavior. *Academy of Management Review* **31(2)**, 386–408.

Karlsson E, Andersson L and Leion P (2000) Daily build and feature development in large distributed projects. In *Proceedings of the International Conference on Software Engineering*, pp 649–658, ACM Press Limerick, Ireland.

Klein KJ, Dansereau F and Hall RJ (1994) Levels issues in theory development, data collection, and analysis. *Academy of Management Review* **19(2)**, 195–229.

Klein KJ and Kozlowski SWJ (2000) From micro to meso: critical steps in conceptualizing and conducting multilevel research. *Organizational Research Methods* **3(3)**, 211–236.

Kraut R and Streeter L (1995) Coordination in software development. *Communications of the ACM* **38(3)**, 69–81.

Larman C (2003) *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, Reading, MA.

Lee G and Xia W (2005) The ability of information systems development project teams to respond to business and technology changes: a study of flexibility measures. *European Journal of Information Systems* **14(1)**, 75–92.

Lepine JA, Piccolo RF, Jackson CL, Mathieu JE and Saul JR (2008) A meta-analysis of teamwork processes: tests of a multidimensional model and relationships with team effectiveness criteria. *Personnel Psychology* **61(2)**, 273–307.

Lewis K (2003) Measuring transactive memory systems in the field: scale development and validation. *Journal of Applied Psychology* **88(4)**, 587–604.

Lui K and Chan K (2006) Pair programming productivity: novice-novice vs expert-expert. *International Journal of Human-Computer Studies* **64(9)**, 915–925.

Marks MA, Mathieu JE and Zaccaro SJ (2001) A temporally based framework and taxonomy of team processes. *Academy of Management Review* **26(3)**, 356–376.

Martin RC (2002) *Agile Software Development, Principles, Patterns, and Practices*. Prentice-Hall, Englewood Cliffs, NJ.

Maruping LM, Venkatesh V and Agarwal R (2009) A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research* **20(3)**, in press.

Moreland RL and Myaskovsky L (2000) Exploring the performance benefits of group training: transactive memory or improved communication? *Organizational Behavior and Human Decision Processes* **82(1)**, 117–133.

Morgeson FP and Hofmann DA (1999) The structure and function of collective constructs: implications for multilevel research and theory development. *Academy of Management Review* **24(2)**, 249–265.

Murru O, Deias R and Mugheddu G (2003) Assessing XP at a European Internet company. *IEEE Software* **20(3)**, 37–43.

Nerur S, Mahapatra R and Mangalaraj G (2005) Challenges of migrating to agile methodologies. *Communications of the ACM* **48(5)**, 72–78.

Nonaka IH and Takeuchi HA (1995) *The Knowledge-Creating Company*. Oxford University Press, Oxford, U.K.

North American and European Enterprise Software and Services Survey (2005). *Corporate IT Leads the Second Wave of Agile Adoption*. Forrester Research Inc. pp 1–6.

Nosek J (1998) The case for collaborative programming. *Communications of the ACM* **41(3)**, 105–108.

Orlikowski W and Iacono S (2001) Research commentary: desperately seeking the 'IT' in IT research – a call to theorizing about the IT artifact. *Information Systems Research* **12(2)**, 121–134.

Panko RR (1999) Applying code inspection to spreadsheet testing. *Journal of Management Information Systems* **16(2)**, 159–176.

Perlow L, Gittell J and Katz N (2004) Contextualizing patterns of work group interaction: toward a nested theory of structuration. *Organization Science* **15(5)**, 520–536.

Pikkarainen M, Wang X and Conboy K (2007) Agile practices in use from an innovation assimilation perspective: a multiple case study. In *Proceedings of the 28th International Conference on Information Systems*, pp 1–17, Association for Information Systems Press, Montreal, Canada.

Podsakoff PM, MacKenzie SB, Lee JY and Podsakoff NP (2003) Common method biases in behavioral research: a critical review of the literature and recommended remedies. *Journal of Applied Psychology* **88(5)**, 879–903.

Poole C and Huisman JW (2001) Using extreme programming in a maintenance environment. *IEEE Software* **18(6)**, 42–50.

Poppendeick M (2001) Lean programming. *Software Development* **9(5)**, 71–75.

Reagans R and McEvily B (2003) Network structure and knowledge transfer: the effects of cohesion and range. *Administrative Science Quarterly* **48(2)**, 240–267.

Ren Y, Carley KM and Argote L (2006) The contingent effects of transactive memory: when is it more beneficial to know what others know? *Management Science* **52(5)**, 671–682.

Rico DF (2008) Effects of agile methods on website quality for electronic commerce. In *Proceedings of the 41st Annual Hawaii International Conference on System Science*, p 464, IEEE Computer Society Press, Waikaloa, Big Island, Hawaii.

Rising L and Janoff NS (2000) The Scrum software development process for small teams. *IEEE Software* **17(4)**, 26–32.

Salas E, Sims DE and Burke CS (2005) Is there a 'Big Five' in teamwork? *Small Group Research* **36(5)**, 555–599.

Schuh P (2001) Recovery, redemption, and extreme programming. *IEEE Software* **18(6)**, 33–40.

Schwaber K and Beedle M (2002) *Agile Software Development with Scrum*. Prentice-Hall, Upper Saddle River, NJ.

Standish Group (2003) The Standish Group report.

Straub DW (1989) Validating instruments in MIS research. *MIS Quarterly* **13(2)**, 147–169.

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

Turk D, France R and Rumpe B (2005) Assumptions underlying agile software development process. *Journal of Database Management* **16(4)**, 62–87.

Walz DB, Elam JJ and Curtis B (1993) Inside a software design team: knowledge acquisition, sharing, and integration. *Communications of the ACM* **36(10)**, 63–77.

Wegner DM (1986) Transactive memory: a contemporary analysis of the group mind. In *Theories of Group Behavior* (Mullen B and Goethals GR, Eds), pp 185–205, Springer-Verlag, New York.

Wegner DM (1995) A computer network model of human transactive memory. *Social Cognition* **13(3)**, 319–339.

Wegner DM, Erber and Raymond P (1991) Transactive memory in close relationships. *Journal of Personality and Social Psychology* **61(6)**, 923–929.

Williams L and Kessler R (2000) All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM* **43(5)**, 109–114.

Williams L and Kessler R (2001) Experimenting with industry's 'pair-programming' model in the computer science classroom. *Computer Science Education* **11(1)**, 7–20.

Wood J and Silver D (1989) *Joint Application Design: How to Design Quality Systems in 40% Less Time*. Wiley, New York.

Appendix

Scales for XP practices

All items use a 7-point Likert scale using 'strongly disagree' and 'strongly agree' as anchors.

Collective ownership

1. Anyone on this team can change existing code at any time.
2. A unit of code can only be changed by the individual who developed it. (*r*)
3. Members of this team feel comfortable changing any part of the existing code at any time.
4. We all feel a sense of responsibility for the system code.
5. If anyone wants to change a piece of code, they need the permission of the individual(s) that coded it. (*r*)

Coding standards

1. We have a set of agreed upon coding standards in this team.
2. Members of this team have a shared understanding of how code is to be written.
3. Everyone on this team uses their own standards for coding. (*r*)
4. We *do not* follow an agreed upon standard for coding software for this project. (*r*)
5. All developers on this team code software according to *their own* set of standards. (*r*)

Note: (*r*) denotes a reverse coded item.