# Coordination In Large Agile Projects

Peng Xu, University of Massachusetts Boston, USA

**ABSTRACT**

*Faced with rapid changes in technology and business environments, more and more information technology (IT) practitioners and researchers are advocating agile methods, which aim to increase customer satisfaction, eliminate waste, accelerate the development process, and lower defects rates. Agile methods, which initially were aimed at small projects, face several challenges when applied to large software projects, however. Concentrating on the challenges of coordinating large agile projects, this study identifies three dimensions of coordination--decision-making structure, communication, and control—and proposes a research framework and a set of propositions to address coordination challenges in large agile projects. Three published case studies are used to illustrate and strengthen the propositions.*

**Keywords:** agile methods, coordination, software development methodology

## 1 INTRODUCTION

Facing rapid changes in technology and business environments, more and more IT practitioners and researchers are advocating agile methods, such as Extreme Programming (XP) and Scrum, as the new generation of software development methodologies. These methods aim to increase customer satisfaction, eliminate waste, accelerate the development process, and lower defects rates (Boehm & Turner, 2003). Both practices and principles of agile methods have been proposed to guide software development. While practices proposed in different agile methods vary, they share such common characteristics as iterative processes, incremental systems development, self-organizing teams, emergent technologies and requirements, dynamic interactions and communications, and reduction of resource-intensive intermediate artifacts (Lindvall et al., 2002; Meso & Jain, 2006).

Initially, most agile methods were aimed at small, non-mission critical projects (Highsmith & Cockburn, 2001). However, more and more large, mission-critical projects have started to explore the possibility of adopting agile methods (Drobka, Noftz, & Raghu, 2004; Fitzgerald, Hartnett, & Conboy, 2006). Prior research demonstrates that though it is not practical to adopt agile practices in their original form, it is reasonable to bring some agility to large, complex projects (Boehm & Turner, 2003; Fitzgerald et al., 2006). Unfortunately adopting processes are not implemented without drama. Numerous difficulties were reported (Fitzgerald et al., 2006; Schalliol, 2001; Taber & Fowler, 2000).

This study concentrates on one of the challenges faced in large software projects that try to adopt agile methods, i.e., coordination. Effective coordination is critical for software development regardless of development methods used (Kraut & Streeter, 1995). Agile methods advocate coordination strategies that are dramatically different from those used in plan-driven methods ( Boehm, 2003; Boehm & Turner, 2003). These coordination strategies work well in small projects, but face problems when applied to large projects (Elssamadisy, 2001; Schalliol, 2001). For example, though face-to-face communication has proven very effective in small projects, it can cause a huge overhead in large teams (Xu & Ramesh, 2007). Large projects need to balance structure and agility when choosing coordination mechanisms.

Prior research on coordination mainly has been conducted in the context of plan-driven methods and fairly stable business environments (Kraut & Streeter, 1995; Levina, 2005; Nidumolu, 1995). Few studies have been done to understand coordination strategies used in agile software development, especially in the setting of large software projects. Motivated by this observation, this study investigates the following research questions:

1. What coordination strategies are available, and how they can help software development in agile methods?
2. How can these coordination strategies be applied to achieve agility in large projects?

This paper addresses these questions by applying coordination theory and developing propositions. To illustrate the arguments more concretely, this paper examines three published case studies on three projects.

The paper is organized as follows. Section 2 discusses coordination strategies used in agile methods and the challenges of applying these strategies in large projects. Section 3 develops a research framework and a set of propositions of coordination in large agile projects. Section 4 illustrates the proposed coordination framework using published case studies. This is followed by a discussion and conclusions.

## 2 COORDINATION IN AGILE METHODS AND CHALLENGES IN LARGE PROJECTS

### 2.1 Coordination in Agile Methods

Different from plan-driven methods, agile methods encourage face-to-face communication, participation of team members in decision-making processes, and shared ownership of artifacts among team members (Cockburn & Highsmith, 2001). For example, agile methods promote daily, short, stand-up meetings instead of long, formal meetings. Agile methods rely on interpersonal interactions to share information and discuss issues instead of using formal documents as a way to convey messages. Agile methods encourage team members to voluntarily sign up for tasks and share ownership instead of assigning tasks from the top-down[1]. The main coordination mechanisms proposed in agile practices are (using practices of XP as examples):

- Daily stand-up meetings
- Co-located teams
- Collective code ownership
- Pair programming
- On-site customers
- Release planning & iteration planning
- Coding standards

### 2.2 Coordination Challenges in Large Projects

Most coordination practices proposed by agile methods emphasize an informal management style. When the project is small, close interactions among team members are effective and problems can be quickly spotted and corrected. However, as the size of the project increases, opportunities for close interactions among project team members drop (Van de Ven, Delbecq, & Koenig Jr, 1976). In large projects, it is difficult for developers to make important decisions only through informal conversations. Miscommunications and misunderstandings happen more often and are more difficult to solve. Large projects need to address unique challenges, such as the knowledge loss caused by turnover of team members and long project duration, complex requirements and interdependency of tasks, and limited resources (Xu & Ramesh, 2007). Relying only on informal strategies is no longer adequate. To summarize, the coordination challenges of using agile methods in large projects are:

- Lack of interaction among participants
- Communication difficulties
- Loss of knowledge
- Complex and unstable requirements
- Complex interdependency tasks
- Technical complexity

[1] http://www.extremeprogramming.org/

Facing these problems and issues, coordination strategies in large projects need to balance agility and discipline (B. Boehm, 2003). Prior studies on coordination in the context of software development are spotty and fragmented (Crowston & Kammerer, 1998; Espinosa, Slaughter, Kraut, & Herbsleb, 2007; Faraj & Sproull, 2000; Wagstrom & Herbsleb, 2006). Most of these studies are conducted in the context of plan-driven methodologies, not in the context of agile methods. None of them examines coordination in large agile projects.

Given the unique challenges faced in coordinating large projects, there is a need to develop a new research framework to help us understand coordination strategies in large agile projects. This provides motivation for the present study.

## 3 COORDINATION IN LARGE AGILE PROJECTS

This section defines coordination in software development, identifies three key aspects of coordination, and develops propositions.

### 3.1 Definition of Coordination

Effective coordination is critical for any team work, especially in software development (Faraj & Sproull, 2000). Several definitions covering different aspects of coordination have been proposed in literature. One research stream focuses mainly on task interdependence in coordination. These studies define coordination as the process of managing dependencies among activities (Crowston & Kammerer, 1998; Malone & Crowston, 1994; Wagstrom & Herbsleb, 2006). This definition recognizes only tasks and tangible resource dependencies in coordination, ignoring its social aspects.

The other research stream on coordination expands the definition to incorporate social interactions among participants. For example, Faraj and Sproull (Faraj & Sproull, 2000) define coordination as team-situated interactions aimed at managing resources and expertise dependencies. This definition includes administrative coordination that manages tangible and economic resource dependencies and expertise coordination that manages knowledge and skill dependencies. Andres and Zmud (Andres & Zmud, 2002) and Espinosa et al. (Espinosa et al., 2007) also adopt this definition and emphasize organic coordination, which relies primarily on informal mechanisms to deal with the social aspects of coordination.

This study adopts the definition of coordination from the second research stream because coordination in software development involves not only task interdependency, but also information and expertise interaction. In this study, coordination is defined as team efforts toward achieving common and explicitly recognized goals and the integration of different parts of teams to accomplish a collective set of tasks (Kraut & Streeter, 1995).

### 3.2 Three Dimensions of Coordination and Their Impacts on Project Performance

Coordination is a multidimensional concept. However, few studies explicitly define the dimensions of coordination. Drawing from prior literature on organization coordination and software development, this study identifies three dimensions of coordination in the context of software development: decision-making structure, communication mode, and control mechanisms. This section discusses each of these dimensions in the context of large agile projects, their implementation, and their impacts on project performance.

#### 3.2.1 Decision-Making Structure

Decision-making is an important activity in coordinating software development in that decision-making structure defines hierarchies, creates decision-making autonomy, and links pins, teams, direct contacts, etc. for a project (Andres & Zmud, 2002). The objective of decision-making structure is to facilitate information flow within and between teams, integrate differentiated tasks and knowledge, solve conflicts, move the project forward, and achieve common outputs.

In software development, stakeholders typically hold conflicting interests and inconsistent requirements. As prior researchers have pointed out, it is important to have appropriate decision-making structures in place that match the project's tasks and social context to address these challenges in coordination (Andres & Zmud, 2002).

There are two types of decision-making structures in software development: decentralization and centralization. *Decentralization* refers to the process by which "unit members are permitted to choose the means for completing the task" (Kyu Kim & Umanath, 1992/1993) (p. 163). In such a structure, decision-making is dispersed in teams; every member is encouraged to actively participate in the decision-making process; they can determine solutions without reporting to higher authorities. This structure is encouraged in agile methods.

In contrast, *centralization* refers to a high degree of organization, a uniform treatment of problems, and a focus on orderliness (Kyu Kim & Umanath, 1992/1993). Such a design is widely used in plan-driven development processes. When adopting this structure, only dedicated authorities are actively involved in the decision-making process, while others are mainly order recipients. Agile methods typically seek to replace centralization with decentralization.

This study proposes that centralization in decision-making is necessary in large agile projects.

*Proposition 1 (P1). Centralization in the decision-making structure has a positive impact on project performance in large agile projects.*

The logical reasoning behind this proposition is as follows. Agile methods faithfully adopt the principle of decentralization that empowers each team members in decision-making and task execution. Decentralization builds up a highly connected network for information flows among peers. Such a structure enhances the team's agility because it increases opportunities for feedback and dynamic adjustments to changes, addressing problems of high uncertainty, such as changing business requirements. For example, XP argues for self-organizing and self-direction instead of rigid pre-planning. These practices have proved effective when dealing with changes in small and midsize projects (Highsmith & Cockburn, 2001; Williams & Cockburn, 2003).

However, large projects that try to adopt agile methods face dilemmas when using a decentralized structure. Decentralization matches the context of small projects that can easily build up a cooperative culture where conflicts can be quickly resolved and agreement can be easily reached among peers. When the team size increases beyond a certain point, decentralization can cause confusion among team members because of the larger volume of information and more complex interdependence among team members and tasks. It is reasonable to argue that in large teams, self-organizing alone is not sufficient in decision-making. Such a mismatch between decision structure and the project needs will lead to delays, chaos, and miscommunication (Kyu Kim & Umanath, 1992/1993). Certain centralization mechanisms are necessary to address these problems. Such centralization includes two specific designs: division of teams and hierarchy of authority in decision-making across teams.

*Proposition 1a (P1a). Division of teams has a positive impact on project performance in large agile projects*

*Proposition 1b (P1b). Hierarchy of authority across teams has a positive impact on project performance in large agile projects.*

These two centralization strategies need to be implemented at the same time to ensure the project's success. The logical reasoning behind this set of propositions is based on organization theory (Hatch, 1997). Each worker has his or her responsibility in a business. When a business grows to a critical size, jobs are grouped into organizational units, such as departments or divisions. Integration of divisions' work makes it possible to achieve the business's goal. Similarly, in software development, to mitigate the risks caused by size and complexity, while also maintaining agility, sub-teams can be held responsible for different parts of the project (Lindstrom & Jeffries, 2004). In such a design, each team can maintain its small size, thus maintaining agility within it.

With multiple teams working on different parts of the system simultaneously, it is important to establish a centralized decision-making unit, such as a project management office that can coordinate efforts across teams to accomplish common project goals (Lindstrom & Jeffries, 2004; Lindvall et al., 2002; Taber & Fowler, 2000). Such legitimate hierarchy is central to large-size projects that want to use agile methods. The main function of this management unit is to achieve concerted action across teams, not to manage the details of each team. Through such a structural arrangement, it is easier to negotiate conflicting interests, reach consensus among team members, forge an effective information network, synchronize goals and efforts across numerous stakeholders, and effectively direct activities in a large project, thus improving overall project performance (Hatch, 1997; Lindvall et al., 2002; Schalliol, 2001; Xu & Ramesh, 2007).

At the same time, within each team that maintains its small size, decentralization should play a central role. I propose:

*Proposition 2 (P2). Decentralization in the decision-making structure within each team has a positive impact on project performance in large agile projects.*

The logical reasoning behind this proposition is straightforward. One of the agile principles is to empower individual members to make decisions effectively without going through a hierarchy, thus making it possible to address problems more quickly (Williams & Cockburn, 2003). To benefit from this principle, it is important for each team to remain small, so that it can fully implement a decentralized decision-making structure within itself.

A centralized decision-making unit is meant to coordinate critical decisions across teams and solve conflicts and discrepancies among teams. It is not meant to intervene with day-to-day operational decisions on the floor. Each team obtains tasks, project goals, directions, and suggestions from the centralized unit and works on its part of the system following typical agile practices. They can use a decentralized structure and empower members to make their own decisions unless such decisions conflict with the overall goals or involve other teams.

### 3.2.2 Communication Mode

*Vertical communication vs. horizontal communication*

In software development, various stakeholders who are in charge of different tasks and possess different expertise need to agree on a common definition of what they are building, so that they can effectively share information and adjust their activities accordingly (Kraut & Streeter, 1995). Communication plays a critical role in this process. Team members need to coordinate by generating feedback through communication (Espinosa et al., 2007).

Communication modes can be differentiated based on their media and extent of formality. One way to differentiate communication modes is by the direction of information flow, i.e., *vertical communication* and *horizontal communication* (Nidumolu, 1995). *Vertical communication* typically involves authorities in the process of information sharing (Nidumolu, 1995). In this mode, information flows from subordinates to their supervisors, who process information and/or forward it to relevant recipients. An example of vertical communication is a team member reporting to the project manager. In contrast, *horizontal communication* occurs mainly through peer-oriented information exchanging, such as daily conversations and sharing artifacts between peers (Nidumolu, 1995).

*Proposition 3 (P3). Horizontal communications within each team has a positive impact on project performance in large agile projects.*

Horizontal communication happens through mutual adjustments and direct interaction among peers. The importance of horizontal communication has been recognized in plan-driven processes (Nidumolu, 1995; Xu & Ramesh, 2007). One significant difference between agile methods and traditional methods is that agile methods regard such communication as a primary mechanism, while plan-driven processes see it as secondary (Boehm & Turner, 2003). In agile projects, intensive peer-to-peer communication is a key success factor. Co-located developers frequently talk to one another face-to-face to solve problems and conflicts. Collective ownership and trust fostered within the team enable developers to rely on horizontal communication without reporting to a higher authority. A simple project design and minimal documents can be easily shared among peers. Such horizontal communication can remove the bottleneck of communication and increase the speed of problem solving and responses.

In large projects, multiple small teams are formed. With a manageable size, each team can adopt the horizontal communication proposed in agile methods as its main communication strategy to increase its agility and performance.

*Proposition 4 (P4). Vertical communication facilitated by the boundary spanners of each team has a positive impact on project performance in large agile projects.*

Vertical communication is associated with hierarchy and social structure (Hatch, 1997). Systems theory suggests that different units in one organization that deal with particular tasks or environments usually establish their own norms, values, time frames, and coding schemes to effectively share and process information (Tushman, 1977). This theory can be extended to a large agile software development project that involves multiple teams dealing with different tasks. Each team has unique problems, environments, stakeholders, interests, and even practices. Such differences can impede communication, hindering the flow of information among teams.

Though when adopting agile methods, it is important to encourage more direct peer interactions as an effective way to share information, communication between teams in large projects needs to incorporate a certain degree of vertical communication. First, stakeholders in software projects often have conflicting interests and goals (Levina & Vaast, 2006). This problem worsens as the size of the project increases. Second, individuals on each team may not have complete information regarding the project because of its size and complexity. In this case, mainly relying on horizontal communication at the individual level between two teams can result in incomplete, ambiguous, and even inconsistent information and knowledge, creating coordination obstacles for the project.

Prior studies highlight the importance of the roles played by boundary spanners in dealing with the challenges of managing across boundaries (Levina, 2005; Levina & Vaast, 2006). Boundary spanners are individuals who are capable of connecting teams and/or work units, obtaining external information, and channeling this information to their colleagues (Tushman, 1977). Such roles help overcome difficulties in gathering and diffusing information across unit boundaries, transform the local settings if necessary to accommodate the counterparts' interests, negotiate for the unit, facilitate sharing, and exchange and combine work produced by separate units (Levina, 2005; Tushman, 1977). Boundary spanners play a gate-keeping role, ensuring the accuracy, completeness, and relevancy of incoming and outgoing information.

In a large project that tries to implement agile methods, it is necessary to appoint key contact persons on each team as boundary spanners to manage the information flow across the team's boundaries. This spanner role needs to be involved in the centralized management unit, which acts as an agent that unites different teams in their pursuit of common project interests. Boundary spanners can use two types of practices to coordinate among teams: community-like and market-like (Levina & Vaast, 2006). In community-like boundary-spanning practices, the spanner uses interpersonal relationships to engage in the project's joint production and negotiation with spanners from other teams. In market-like practices, the spanner plays an intermediary role by presenting the team-work product and facilitating information sharing and discussions among teams.

*Personal Communication vs. Impersonal Communication*

The second way to differentiate between communicate modes is by formality. Extensive literature on knowledge sharing has discussed two important knowledge types based on the sharing media: explicit knowledge and tacit knowledge. The former refers to knowledge that can be codified and stored in certain media, such as documents and databases, while the latter refers to knowledge that people carry in their minds, which is, therefore, difficult to access (Nonaka, 1994). Explicit knowledge and information usually are associated with *impersonal communication*, which relies on media such as documents and reports for sharing, while tacit knowledge is associated with *personal communication*, which relies on interpersonal interaction. As crucial strategies of coordination, both communication modes are deployed in software development (Tiwana & Mclean, 2005; Xu & Ramesh, 2007). Managers need to balance these two according to the project's context and development method.

Agile methods promote personal over impersonal communication, arguing that by doing so, projects can remove much overhead and increase speed. However, in large agile projects, personal communication needs to be supplemented by impersonal communication.

*Proposition 5 (P5). Personal communication that is supplemented by impersonal communication (boundary objects) has a positive impact on project performance.*

Individual interaction is more important than processes and documents in agile methods (Highsmith & Cockburn, 2001). Based on this principle, multiple practices, such as iteration meetings, daily stand-up meetings, and on-site customers have been proposed for personal communication. Social interaction (personal communication), which has proven effective in small and midsize projects (Cockburn & Highsmith, 2001), is valued more than impersonal communication in agile methods.

Though personal communication that primarily relies on tacit knowledge can improve agility and reduce communication costs in small projects, the effectiveness of this communication mode decreases as the size of project increases (Xu & Ramesh, 2007). A large team faces difficulties in sharing relevant project information with a large number of participants over time. A prerequisite for effective personal communication is a cohesive, trusting culture within the team. Such a culture is difficult to build in large projects that involve numerous stakeholders. This obstacle hinders effective personal communication, thus affecting project performance (Boehm & Turner, 2003). Therefore, though it is important to leverage the benefits of personal communication as proposed by agile methods, large agile projects also need to supplement it with a more formal communication mode, i.e., impersonal communication.

Impersonal communication uses artifacts as means of sharing and preserving information and knowledge. These artifacts, which can be as formal as system architecture or as informal as post-it notes, can be fully incorporated into the team's daily practices and used to preserve knowledge and document critical decisions.

Such objects are even more important in communication between teams. Prior studies have recognized the importance of boundary spanning that can effectively connect internal units (Levina & Vaast, 2006; Tushman, 1977). In large agile projects, the important role of boundary spanners needs to be supplemented with appropriate boundary spanning objects. The concept of boundary objects is introduced to address the limitations of boundary spanners (Levina & Vaast, 2006). Boundary spanners, connecting different units, may have limited social networks or face temporal and physical constraints that prevent them from effectively directing communication across boundaries (Tushman, 1977). In such cases, *boundary objects,* defined as a broad range of artifacts that can reflect local needs and interests, can bridge the communication gap between groups separated by location, hierarchy, and functions (Levina & Vaast, 2005). Examples of boundary objects in the context of software development are prototypes, design models, and standardized reporting forms. These objects reflect practices and knowledge from each team and represent teams' work and views (Levina & Vaast, 2005). They help boundary spanners and the centralized management unit collect information, continue discussion, and integrate knowledge across teams. These objects can provide the basis for coordination and negotiation among teams, thus positively affecting project performance.

### 3.2.3 Control

As the third dimension of coordination in software development, control attempts to ensure that individuals act according to agreed-upon strategies to achieve desired objectives by fusing together complementary roles and motivating individuals to work in accordance with organizational goals and objectives (Kirsch, 1997). According to coordination theory, control mechanisms must facilitate the information exchanges and decisional autonomy needed for effective coordination (Andres & Zmud, 2002).

*Informal control*

Prior studies have categorized control strategies as informal and formal (Choudhury & Sabherwal, 2003; Kirsch, 1997; Kirsch, Sambamurthy, Ko, & Purvis, 2002). Informal control is encouraged by agile methods and proposed as the main control mechanisms in agile projects. *Informal control* is mainly based on social or people strategies. It includes *clan control* and *self control* (Kirsch, 1997). Clan control is implemented "by promulgating common values, beliefs, and philosophy within a clan which is defined as a group of individuals who are dependent on one another and who share a set of common goals" (Kirsch, 1997) (p. 217). Examples of clan control include socialization, shared values, trusting culture, and training. Self control acts as a function of individual objectives and intrinsic motivation, and depends on individuals' self-monitoring, self-rewarding, and self-sanctioning ( Kirsch, 1997; Kirsch et al., 2002).

*Proposition 6 (P6). Informal control within each team has a positive impact on project performance in large agile projects.*

Informal control should play a dominant role within each team as long as the team is of manageable size, as proposed in agile methods. Informal control, including clan control and self control, is important, especially at the beginning of the project when outcome measurements are still vague (Kirsch et al., 2002)

Clan control is exercised via peer pressure. The key success factor in clan control is to build up a cohesive culture within the team. Proposed agile practices, such as daily stand-up meetings, collective ownership, and pair programming, if well executed, can all contribute to trust building and clan control. These practices facilitate dynamic interaction among team members so that they can quickly share and understand team values and goals, and at the same time, use common values and goals to guide individual behaviors.

Self control is also an important control mechanism for coordination in agile methods (Kirsch et al., 2002). Self-control mechanisms rely on individuals to establish standards and motivate themselves to work toward project goals (Choudhury & Sabherwal, 2003). The proposed agile practices, such as allowing individuals to voluntarily sign up for tasks, encouraging collective ownership, and empowering individuals by granting autonomy in what individual does on job and how individual does the work, promote effective self-control mechanisms.

Effectively implementing informal control within each team that maintains its small size can motivate each individual and, at the same time, save overhead associated with formal control (Cockburn & Highsmith, 2001).

*Formal control*

Formal control includes two mechanisms: behavioral control and outcome control. *Behavior control* uses specific rules and procedures to ensure desired outcomes, while *outcome control* relies on articulating standards and goals and rewarding team members who meet the goals (Kirsch, 1997; Kirsch et al., 2002). Examples of behavior control in software development are using well-defined development processes and providing specific job descriptions. Examples of outcome control in software development include establishing deadlines and budgets, outlining performance expectations, specifying milestones, and signing contracts with users.

Though formal control does not get strong support from agile methods, in large agile projects, some degree of formal control, especially at the project level (across teams), is necessary for effective project performance.

*Proposition 7 (P7). Formal control across teams has a positive impact on project performance in large agile projects.*

The logical reasoning for this proposition is as follows. Agile methods mainly rely on social processes and promote informal control. However, project size does affect the choice of control modes (Meso & Jain, 2006). While informal control is suitable for small projects, large projects with multiple teams need to add more formal control elements in efforts to coordinate across teams (Kirsch, 1997).

The centralized management unit needs to control across teams by influencing each team's behaviors (behavior control). Unlike in plan-driven software projects, where behavior control is implemented by well-defined rules, processes, and team hierarchy, the centralized management unit does not need to specify details of methods and processes for each team. Instead, it only needs to explicitly define and assign tasks to different teams as part of its behavior control efforts. The management unit does not directly observe and influence every single developer's behavior, but exercises its behavioral control by influencing the contact person of each team and observing the various teams' aggregated work.

Agile methods emphasize system evolution, which "allows the solution being developed to be responsive to the emerging changes in project requirements by taking into account feedback gained from the exercise of frequent releases and integration" (Meso & Jain, 2006)(p. 23). However, facing multiple teams that are in charge of different parts of the system, the centralized management unit needs to explicitly specify the expected outcomes for each team so that it can coordinate teams' efforts and integrate their work. Formal control mechanisms at the project level are crucial to overall project performance.

## 4 APPLYING THE RESEARCH FRAMEWORK TO EMPIRICAL SETTINGS

To illustrate the arguments made in this paper and the proposed research framework, this section briefly examines three published cases. All three cases are large software development projects that tried to adopt agile methods. These cases are chosen because they show how large, complex projects can embrace agile practices in their coordination efforts. The purpose is to provide examples to strengthen the research propositions more concretely and show how propositions can be applied in real world examples. They cannot be considered as empirical data to validate the claims. However, it is instructive to examine how the proposed research propositions can be applied to empirical settings. Prior studies have used this method in studies of organization theory development (Overby, 2008) and software development (Lyytinen & Robey, 1999).

### 4.1 Cases Background

Case 1: ThoughtWorks

ThoughtWorks is a Chicago-based system integration and consulting company specializing in building business applications. The case reported is a leasing application. After spending about 18 months following plan-driven methods, the project encountered numerous serious problems. The project then introduced Extreme Programming (XP) as a new method, hoping that the agile methods could help solve problems faced by the project, which had a team of about 50 people. Three papers examine this case study (Elssamadisy, 2001; Schalliol, 2001; Taber & Fowler, 2000). Each paper approaches this case from a different perspective. Therefore, they provide a fairly comprehensive picture of the project.

Case 2: Intel Shannon

This case study examines Intel Shannon of Ireland, which is Intel's Infrastructure Processor Division and employs about employs 125 people, among which 90 are involved in software development (Fitzgerald et al., 2006). The study at Intel Shannon is based on the software development of two product families, the IXP2XX and IXP4XX network processors. The IXP2XX project involved approximately 15 engineers, lasting 18 months, and the IXP4XX product had over 30 engineers, lasting about 24 months. As a CMM level 2 organization, Intel Shannon decided to adopt XP to cope with time-to-market pressure. Intel Shannon had used agile method for five years before this study was conducted and had committed and experienced developers.

Case 3: Radio System-A Mission Critical Project

This case describes a complex mission-critical, two-way radio system (Drobka et al., 2004). The project, which lasted 18 months, is part of a large system. The ship dates are set well before the customers finalize their product requirements.

## 4.2 Coordination Strategy in Three Cases

### 4.2.1 Decision-making Structure

*Centralization (P1): Division of Teams (P1a) and Hierarchy of Authority across Teams (P1b)*

Proposition 1 (a & b) proposes that large teams need to divide into small teams and establish a hierarchy of authority across teams. These three cases demonstrate how it works.

About 50 people work on the ThoughtWorks project. Twenty-five developers are divided into two teams: the domain team and the testing team. Each team maintains a small size (about 12 members). The domain team is solely responsible for new functionality, and the testing team is responsible for testing and fixing bugs. There is also a quality assurance (QA) team and an analyst team, each consisting of about eight members respectively. The project establishes a project management office that consists of two account managers who serve as main contact with clients, two iteration managers who manage two teams respectively, and a release plan manager who facilitates and makes the decisions on how features written in story cards will play in the long-term plan. The management office plays the authority role, managing coordination at the project level and coordinating efforts across the teams.

Similarly, at Intel Shannon, the IXP2XX project involves approximately 15 engineers split into four teams across three sites, and the IXP4XX project consists of five teams and over 30 engineers, across two sites. Each team has team lead who manages the daily tasks of each team. There is also an overall project lead who plays the authority role in decision-making and receives reports from the team leads.

The radio system involves four teams, each of which is in charge of different parts of the system. The project managers working with domain experts prioritize the features required for each release. Any changes to the iteration plan need to be approved by project managers, who have the authority to move functionality from iteration to iteration. In this case, the project managers form the centralized management unit that mainly works on overall release plans and coordinates tasks across the four teams.

All three cases demonstrate how team division and centralization work in their respective decision-making structures.

*Decentralization within Each Team (P2)*

Though all three projects use a hierarchical structure, they all maintain decentralization within each team to embrace the spirit of agility.

In the radio system, after the project managers prioritize functionality and make the release plans, each team holds its own planning meetings. During the meeting, team members assess and discuss the iteration plan together and sign up for tasks voluntarily. Though the project managers recommend rules about what processes and practices need to be followed, each team has the authority to modify these practices when appropriate to suit their needs. For example, instead of allocating a dedicated customer as managers proposed, one team rotated the team's senior members through the role.

At ThoughtWorks, development staff meets as a group to assess and discuss tasks and voluntarily sign up for tasks for the upcoming iteration. Every member participates in this decision-making process. Collective ownership is encouraged. Individuals are given ownership to their own problems. Intel Sharon adopts similar strategies.

In sum, all three cases show that within each team, the decision structure is decentralized to give more power to individuals.

### 4.2.2 Communication

*Horizontal communications within each team (P3)*

All three cases report that they rely on horizontal communications within each team. Teams meet for a quick stand-up meeting to discuss the project's progress and issues every day or every other day. They all use pair-programming for team members to work together and share information when appropriate. This enables team members to frequently engage in conversation with others and share knowledge within the team. Collective ownership goes hand in hand with horizontal communication and provides opportunities for individuals to become exposed to all team tasks. It equips each member with adequate background knowledge and facilitates discussion among team members. Horizontal dissemination of information is also done via rotating team members through different parts of the code. It is reported that very good communications takes place via informal chats, code-reviews, and short stand-up meetings.

*Vertical communication facilitated by boundary spanners (P4)*

Vertical communication also plays an important role in all three cases examined in this paper. Though agile methods promote horizontal communication, these cases demonstrate the importance to leverage benefits offered in vertical communication in large projects, as suggested in proposition 4. In each case, the project designates managers, leads, and/or experts as boundary spanners, who guard and disseminate information across the team boundary.

At ThoughtWorks, multiple clients representing different divisions and units are involved. To manage competing voices, project managers require clients and developers to report critical requests and changes so that managers can facilitate negotiations between clients and developers on decisions regarding how to prioritize development plans and functions. Each team also needs to report important decisions, such as iteration planning, to managers. In this case, managers (two account managers, two iteration managers and release plan manager) serve as boundary spanners who monitor and direct information flows across teams.

Intel Shannon also finds that horizontal communication across multiple teams is not realistic because of the overhead at a large scale. To improve communication, it incorporates vertical communication mechanisms. After each team discusses iteration plans, team leads outline the final plan and report the milestones and the contents of each iteration to the overall project lead, who integrates information and informs team leads. In this case, team leads and the project lead serve as the boundary spanners.

During the radio system project, each team designates domain experts as the customer proxy. The experts serve as boundary spanners, who are responsible for gathering information about the system's requirements. Communication between developers and clients occurs through the customer proxy. These spanners are also responsible for interfacing with other teams and coordinate communications between teams.

In sum, all three cases adopt vertical communication and boundary spanners and show that such strategies work well in large agile projects.

*Personal Communication Supplemented by Impersonal Communication (Boundary Objects) (P5)*

Agile methods promote personal communication and argue that personal communication can replace intermediate artifacts. However, personal communication can be constrained in large projects because of the extensive information and complexity in team structure. As indicated in proposition 5, personal communication needs to be supplemented by impersonal communication (e.g., boundary objects).

At ThoughtWorks, teams rely heavily on verbal communication most of the time, as proposed in agile methods. As the system grows in size and complexity, teams have difficult time keeping track of progress because there is no holistic picture of the application available to everyone during the development process. Individuals know local functions, but fail to grasp the connections in their minds because of the large scope. All of these hinder discussions and sharing among team members. The situation becomes worse when new members join the teams. Therefore, the teams draft a sketch of the architecture design (a boundary object) to maintain the big picture of the system. Such object helps teams see the connections of the different parts of the system and facilitate communication between teams. ThoughtWorks also uses demos and narratives (less formal use cases) for team members to understand the system.

Intel Shannon also incorporates impersonal communication and uses several boundary objects. In addition to personal communication, a simple design document is created before different teams take on coding. This design document is shared by all teams as a guideline and is the basis for release and iteration planning. During short daily meetings, team members use post-it notes, an informal boundary object, to make their ideas and decisions visible. Team members are encouraged to bring their next 24-hour plans before each meeting and post them in their named area. The completed tasks are moved to "done" area. This helps team members in tracking progress, visualizing the whole system, preparing more thoroughly before daily meetings, and engaging in productive discussion on the project.

In a similar vein, the radio system project developed a high-level architecture document to provide a roadmap for developers. To further facilitate communication within and between teams, artifacts (boundary objects) with common formats, such as use cases, are created and shared among teams. Such boundary objects provide a common language and knowledge for communication within and across teams.

In all three cases, impersonal communication and boundary objects are proved necessary in large agile projects for more effective communication. However, they by no means can replace the primary role played by personal communication.

### 4.2.3 Control

*Informal Control within Each Team (P6)*

Proposition 6 proposes that each team needs to follow the informal control proposed by agile methods to maximize benefits of agility. Both ThoughtWorks and Intel Sharon use this strategy. They adopt pair programming, a clan control mechanism to ensure code quality. Pair programming has proved to save considerable debugging time. It is reported in the case of Intel Sharon that one of the teams that used pair programming achieved zero-defect quality, while another that did not use pair programming, had the highest defect density. Developers who are involved in pair programming report higher job satisfaction and are more enthusiastic about their work compared to those who do not use pair programming, according to the case of Intel Sharon. In this way, pair programming helps build trust and a cooperative culture, which further motivates self-control. Both ThoughtWorks and Intel Sharon encourage team members to voluntarily sign up for tasks and give them ownership of the tasks, a strategy that motivates both clan and self control.

*Formal control across teams (P7)*

Though agile methods encourage informal control, certain formal control mechanisms are proved necessary in large agile projects. For example, at ThoughtWorks, user requirements, detailed in a narrative document, must be approved and signed by clients. Also, upon finishing their tasks, team members must obtain clients' formal approval. This formal approving process helps control project outcomes and ensures consistency in large projects. Intel Shannon implemented formal control in the form of a strict coding standard defined early in the project and referred to it extensively during the project as a way to control the coding processes and outcomes.

The radio system also takes steps in formal control to ensure quality. Critical documents (e.g., use cases) and changes need to go through a formal review process to get approval. Such behavioral control ensures that everything is consistent with the overall system requirements and plan. To further strengthen consistency and quality across teams, the project establishes a formal review process at the end of each iteration for all four teams. External experts are invited to assess the design and code and point out errors and inefficiency. All three cases demonstrate the importance of formal control in large agile projects.

## 5 DISCUSSIONS AND CONCLUSIONS

Drawing from the literature and published case studies, this study examined three dimensions of coordination in large software projects that embrace agile methods. Table 1 summarizes how large agile projects coordinate.

**Table 1. Summary of Coordination Strategy in Large Agile Projects**

| Dimensions of Coordinating Large Agile Projects | Classification | Strategies |
|---|---|---|
| Decision-making structure | Centralization | Divide the project into multiple teams, each of which maintains small or mid size and is responsible for a subset of the tasks; Establish a centralized management unit that allocates resources, solves conflicts and coordinates efforts across teams |
| | Decentralization | Use decentralization structure as proposed in agile methods within each small team. |
| Communication | Vertical | Rely on vertical communication channel across teams |
| | Horizontal | Use horizontal communication channel within each team as proposed in agile methods |
| | Personal | Rely on personal communication within each team supplemented by impersonal communication |
| | Impersonal | Rely on impersonal communication using boundary objects between teams |
| Control | Formal | Use formal control across teams |
| | Informal | Use informal control within each team |

This study suggests the choices of coordination strategies in the context of agile methods are contingent on project size. Consistent with prior research (Boehm, 2003), this study shows that it is necessary to maintain the team size to fully embrace the practices of agile methods. As the project size increases, more formal coordination strategies, such as centralization, vertical communication, impersonal communication, and formal control are necessary. Original agile practices are applicable within each team that remains small to midsized. However, at the project level, across multiple teams, more formal coordination strategies are needed because of the complexity of the large project and differences among teams. This can embrace the spirit of agile methods, but at the same time maintain the balance between agility and discipline that is deemed critical in large projects (Boehm, 2003). The three cases cited in this paper by no means validate the study's propositions, but they do provide concrete evidence. At the same time, these cases demonstrate how each proposition can be implemented in real project contexts.

This study contributes to both the literature of agile methods and agile practices. Though more and more attention has been paid to agile methods and their applications in various contexts, no prior studies have examined coordination strategies in agile projects, especially in large agile projects. By examining the literature and published case studies, this study identifies three dimensions of coordination in software development and discusses how different communication strategies, decision-making structures, and control modes help large projects become agile and improve project performance. Though a few papers and books focus on balancing agility and discipline (Boehm, 2003; Boehm & Turner, 2003), none of them explicitly explain it from the perspective of coordination. The present research addresses this gap. This study also provides insights for practitioners, especially project managers that manage agile projects. The study suggests how to choose and apply various coordination strategies at different levels. The specific strategies discussed in this paper are of interest to practitioners.

## AUTHOR INFORMATION

**Peng Xu** is an Assistant Professor at the Department of Management Science and Information Systems of the University of Massachusetts Boston. She received her Ph.D. in Computer Information Systems from Georgia State University in 2004. Her research areas are software process, software engineering, knowledge management, and process agility. Her work has appeared in journals such as the *Journal of Management Information Systems, Communications of the ACM, Requirements Engineering Journal, Information and Management, and Decision Support Systems.*

**REFERENCES**

1. Andres, H. P., & Zmud, R. W. (2002). A Contingency Approach to Software Project Coordination. *Journal of Management Information Systems, 18*(3).
2. Boehm, B. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*: Addison Wesley.
3. Boehm, B., & Turner, R. (2003). Using Risk To Balance Agile And Plan-Driven Methods. *IEEE Computer*, 57-66.
4. Choudhury, V., & Sabherwal, R. (2003). Portfolios of Control in Outsourced Software Development Projects. *Information Systems Research, 14*(3), 291 –314.
5. Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *IEEE Computer, 34*(11), 131-133.
6. Crowston, K., & Kammerer, E. E. (1998). Coordination and Collective Mind in Software Requirements Development. *IBM SYSTEMS JOURNAL, 37*(2), 227-245.
7. Drobka, J., Noftz, D., & Raghu, R. (2004). Piloting XP on Four Mission-Critical Projects *IEEE Software, 21*(6), 70-75.
8. Elssamadisy, A. (2001). *XP On A Large Project – A Developer's View*. Raleigh, NC: XP/Agile Universe.
9. Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007). Team Knowledge and Coordination in Geographically Distributed Software Development. *Journal of Management Information Systems, 24*(1), 135–169.
10. Faraj, S., & Sproull, L. (2000). Coordinating Expertise in Software Development Teams. *Management Science, 46*(12), 1554-1568.
11. Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising Agile Methods to Software Practices at Intel Shannon. *European Journal of Information Systems, 15*(2), 200.
12. Hatch, M. J. (1997). *Organization Theory*. New York: Oxford University Press.
13. Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *IEEE Computer, 34*(9), 120-122.
14. Kirsch, L. J. (1997). Portfolios of control modes and IS project management. *Information Systems Research, 8*(3), 215-239.
15. Kirsch, L. J., Sambamurthy, V., Ko, D.-G., & Purvis, R. L. (2002). Controlling Information Systems Development Projects: The View from the Client. *Management Science, 48*(4), 484–498.
16. Kraut, R. E., & Streeter, L. A. (1995). Coordination in Software Development. *Communications of the ACM, 38*(3), 69-81.
17. Kyu Kim, K., & Umanath, N. (1992/1993). Structure and Perceived Effectiveness of Software Development Subunits: A Task Contingency Analysis. *Journal of Management Information Systems, 9*(3), 157-181.
18. Levina, N. (2005). Collaborating on multiparty information Systems development projects: A collective reflection-in-Action view. *Information Systems Research, 16*(2), 109 –130.
19. Levina, N., & Vaast, E. (2005). The emergence of boundary spanning competence in practice: implications for implementation and use of information systems. *MIS Quarterly, 29*(2), 335-363.
20. Levina, N., & Vaast, E. (2006). Turning a Community into a Market: A Practice Perspective on Information Technology Use in Boundary Spanning. *Journal of Management Information Systems, 22*(4), 13-37.
21. Lindstrom, L., & Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. *Information Systems Management 21*(3), 41-52.
22. Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., et al. (2002). Empirical Findings in Agile Methods/ Agile Universe 2002. Paper presented at the Extreme Programming and Agile Methods.
23. Lyytinen, K., & Robey, D. (1999). Learning Failure in Information Systems Development. *Information Systems Journal, 9*, 85-101.
24. Malone, T. W., & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys, 26*(1), 87-119.
25. Meso, P., & Jain, R. (2006). Agile Software Development: Adaptive Systems Principles And Best Practices. *Information Systems management*, 19-30.
26. Nidumolu, S. (1995). The Effect of Coordination and Uncertainty on Software Project Performance: Residual Performance Risk as An Intervening Variable. *Information Systems Research, 6*(3), 191-219.

27. Nonaka, I. (1994). A Dynamic Theory of Organizational Knowledge Creation. *Organization Science, 5*(1), 14-37.
28. Overby, E. (2008). Process Virtualization Theory and the Impact of Information Technology. *Organization Science, 19*(2), 277-291.
29. Schalliol, G. (2001). *Challenges for Analysts on A Large XP Project*. Raleigh, NC: XP/Agile Universe.
30. Taber, C., & Fowler, M. (2000). An Iteration in the Life of an XP Project. *Cutter IT Journal, 13*(1), 13-21.
31. Tiwana, A., & Mclean, E. R. (2005). Expertise integration and creativity in information systems development. *Journal of Management Information Systems, 22*(1), 13–43.
32. Tushman, M. L. (1977). Special Boundary Roles in the Innovation Process. *Administrative Science Quarterly, 22*(4), 587-605.
33. Van de Ven, A. H., Delbecq, A. L., & Koenig Jr, R. (1976). Determinants of Coordination Modes within Organizations. *American Sociological Review, 41*(April), 322-338.
34. Wagstrom, P., & Herbsleb, J. (2006). Dependency Forecasting in the Distributed Agile Organization. *Communications of the ACM, 49*(10), 55-56.
35. Williams, L., & Cockburn, A. (2003). Agile Software Development: It's about Feedback and Change. *IEEE Computer, 36*(6), 39-43.
36. Xu, P., & Ramesh, B. (2007). Software Process Tailoring: An Empirical Investigation. *Journal of Management Information Systems, 24*(2), 293-328.

**NOTES**