

# Coordination in co-located agile software development projects

Diane E. Strode\*, Sid L. Huff, Beverley Hope, Sebastian Link

School of Information Management, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand

## ARTICLE INFO

### Article history:

Received 4 August 2011

Received in revised form 27 January 2012

Accepted 7 February 2012

Available online 15 February 2012

### Keywords:

Agile methods

Agile software development project

Coordination effectiveness

Coordination strategy

Coordination Theory

Extreme Programming

Scrum

## ABSTRACT

Agile software development provides a way to organise the complex task of multi-participant software development while accommodating constant project change. Agile software development is well accepted in the practitioner community but there is little understanding of how such projects achieve effective coordination, which is known to be critical in successful software projects. A theoretical model of coordination in the agile software development context is presented based on empirical data from three cases of co-located agile software development. Many practices in these projects act as coordination mechanisms, which together form a coordination strategy. Coordination strategy in this context has three components: synchronisation, structure, and boundary spanning. Coordination effectiveness has two components: implicit and explicit. The theoretical model of coordination in agile software development projects proposes that an agile coordination strategy increases coordination effectiveness. This model has application for practitioners who want to select appropriate practices from agile methods to ensure they achieve coordination coverage in their project. For the field of information systems development, this theory contributes to knowledge of coordination and coordination effectiveness in the context of agile software development.

© 2012 Elsevier Inc. All rights reserved.

## 1. Introduction

Agile software development is a new paradigm in information systems development that provides a way to organise complex multi-participant software development while accommodating constant project change. Although this class of system development methodology is accepted practice amongst the practitioner community and has been adopted and adapted in a multitude of different project contexts (Ambler, 2009; Dyba and Dingsoyr, 2008), it is still of intense practical and theoretical interest due to its wide-ranging effects. Adoption of the agile approach impacts organisation structure, culture, and roles, contract negotiation, human resourcing, the way customers are involved in projects, the way system quality is negotiated, and the way that software is delivered. Agile software development also changes many facets of software project practice, both technical practices and team social interaction. This means that existing theories about software development organisations, project management, and system development may need to be re-evaluated to accommodate this new paradigm (Dingsoyr et al., 2010).

Long before the advent of agile software development, effective coordination was acknowledged as a critical element in

organisations generally, and in software development in particular (Curtis et al., 1988; Kraut and Streeter, 1995; Van de Ven et al., 1976). Nidumolu (1995) articulated the problem clearly when he asked, “How can software development projects be coordinated more effectively in the presence of uncertainty?” (p. 213). Agile software development methods were particularly designed to deal with change and uncertainty, yet they de-emphasise traditional coordination mechanisms such as forward planning, extensive documentation, specific coordination roles, contracts, and strict adherence to a pre-specified process. Instead, they favour intensive face-to-face communication and other apparently simple practices. Yet, agile software development has contributed to the success of many software projects, so arguably effective coordination, of some type, is taking place within the agile approach. However, the form and nature of this coordination is not well understood.

It is timely to reinvestigate the important concept of coordination in the light of this new paradigm in information systems development (Agerfalk et al., 2009; Cao and Ramesh, 2007; Dingsoyr et al., 2008; Kautz et al., 2007). A large and diverse body of coordination research exists in contexts relevant to software development practice, but agile software development came into common use subsequent to much of that research. Therefore applying a coordination lens to agile software development may uncover facets of coordination or software development not previously identified.

In any study of coordination, there are two important considerations. The first is to find out what activities and artefacts in a

\* Corresponding author. Tel.: +64 4 472 1000; fax: +64 4 499 4601.

E-mail addresses: [diane.strode@vuw.ac.nz](mailto:diane.strode@vuw.ac.nz) (D.E. Strode), [sid.huff@vuw.ac.nz](mailto:sid.huff@vuw.ac.nz) (S.L. Huff), [beverley.hope@vuw.ac.nz](mailto:beverley.hope@vuw.ac.nz) (B. Hope), [sebastian.link@vuw.ac.nz](mailto:sebastian.link@vuw.ac.nz) (S. Link).

situation support coordinated action, and the second is to identify the characteristics of a highly coordinated state – i.e. what such a state “looks like.” Once these two aspects of coordination are clarified, it becomes possible to investigate which activities and artefacts are more, or less, effective at bringing about a highly coordinated state. The purpose of this article is to identify agile practices that have a coordinative function and how they fit together to form a coordination strategy. In addition, how this coordination strategy contributes to effective coordination is explored. Two related research questions guide the study:

1. How is coordination achieved when using an agile development approach?
2. What is the relationship between coordination strategy and project coordination effectiveness in the context of agile software development?

This article follows the guidelines of Eisenhardt and Graebner (2007) for theory building from positivist multi-case study research where the aim is to define concepts and elaborate their relationships in a form that makes them suitable for testing in future work. The findings come from three cases, each one an independent software project using an agile software development method.

The article is organised as follows. First, a brief review of the current state of research on agile software development provides a background to the study. There follows a discussion of coordination in the literature of organisation studies, information systems project studies, and teamwork, highlighting the lack of a clear framework in that literature for exploring agile software development. Then the small body of research linking coordination with agile software development is discussed. Following this is a discussion of how the concepts of coordination strategy and coordination effectiveness fit within a general framework of project success. A description of the case study research method used in the study is next, followed by the findings, which include a description of each project including its development methodology and coordination problems. Then a theoretical model of coordination in agile software development projects, with supporting evidence from the cases is presented. Propositions linking the coordination strategy and coordination effectiveness concepts of the model are presented followed by a discussion section where the research questions are addressed along with implications of the theory for IS practitioners. The strengths and limitations of the research are examined and the article concludes by considering the ways in which further research in this area can make use of this theory of coordination in an agile software development context.

## 2. Agile software development

For forty years system development methodologies have been designed for different types of organisation, different types of problem, and different technologies (Avison and Fitzgerald, 2006; Oile et al., 1983). In the 1990s, the approach to system development called agile software development first appeared (Iivari et al., 2004). This approach is designed to support flexible, rapid, and effective development under conditions of change, uncertainty, and time pressure (Dyba and Dingsoyr, 2008). These conditions are common in the current fast-paced business environment (Madsen, 2007) where agile software development has proven efficacy (Baskerville et al., 2007). Amongst practitioners the popularity of this approach has grown rapidly since it was first introduced (Ambler, 2009; West and Grant, 2010) and some researchers argue that agile software development has fundamentally changed the practice of software development (Abrahamsson et al., 2009; Agerfalk et al., 2009; Baskerville et al., 2007; Rajlich, 2006).

‘Agile software development’ is a catch-all term which includes a variety of agile methods. All agile methods are founded on a distinct philosophy and a set of practices for conducting information systems development (Abrahamsson et al., 2003; Hilkka et al., 2005). There are at least 13 different agile methods, the most popular are Extreme Programming (XP) and Scrum. Others include Dynamic Systems Development Method (DSDM), Crystal methods, Adaptive System Development, Lean Software Development, Feature Driven Development, and Evo (Dyba and Dingsoyr, 2008). The unifying philosophy of agile methods derives from the ‘agile manifesto’ developed by a number of system development experts who had designed different methodologies based on similar ideals (Beck et al., 2001). This philosophy emphasises the importance of individuals and their interactions, teamwork, production of early working software, collaboration with customers, and responding effectively to change. This is in preference to the traditional focus on process, tools, documentation, contract negotiation, and following plans. Each agile method loosely conforms to this philosophy (Conboy et al., 2005), yet each agile method has a distinct focus (e.g. Scrum for project management, XP for rapidly producing quality software) and an integrated set of work practices and iterative micro-phases.

Before 2005, few rigorous studies of agile software development had been conducted. More recently, prompted in part by a review of research on this topic by Dyba and Dingsoyr (2008), rigorous research methods have been used to explore how agile software development is used. Detailed ethnographic studies have shed light on how practitioners perform agile development (MacKenzie and Monk, 2004; Sharp and Robinson, 2004) and carefully conducted case studies have provided insights into agile development in specific contexts such as health care systems (Fruhling and de Vreede, 2006), software product families (Fitzgerald et al., 2006), and distributed system development (Sarker and Sarker, 2009).

There is a large body of existing theory in organisation studies, management science, and information systems that could be applied to the agile approach to better understand, explain, and predict its effects and efficacy. Cao and Ramesh (2007), Nerur and Balijepally (2007) and Dingsoyr et al. (2008) have called for this type of work. Examples of research of this sort include a study by Conboy (2009), who defined a theory of agility in the context of systems development by linking agile software development to agility in other fields. In addition, empirically supported studies linking distributed agile software development and project success concepts have recently appeared (Sarker et al., 2009). Maruping et al. (2009) used control theory to explain the impact of agile software development on software project quality; Harris et al. (2009) extended control theory to explain the effectiveness of flexible management practices.

Research into the agile approach to software development has matured in the past five years. Nevertheless, a number of open questions remain, and the relevance and implications of certain fundamental organisational concepts are still not fully understood in this context (Abrahamsson et al., 2009; Agerfalk et al., 2009). One such concept is coordination. In the next section, we briefly review previous research on coordination, which is scattered across a number of different domains.

## 3. Previous research on coordination

No single widely accepted definition of coordination exists. When Malone (1988) first proposed a theory of coordination, he defined it in this way: “when multiple actors pursue goals together, they have to do things to organise themselves that a single actor pursuing the same goals would not have to do. We call these extra organising activities coordination” (Malone, 1988, p. 5). Later this definition was refined by Malone and Crowston (1994) when they

developed their interdisciplinary theory of coordination. Their theory is based on the tenet that “*coordination is the managing of dependencies*” (Malone and Crowston, 1994, p. 90). In contrast, knowledge management researchers define coordination as a way to manage “*problems of sharing, integrating, creating, transforming, and transferring knowledge*” (Kotlarsky et al., 2008, p. 96). While there is no consensus on a definition of coordination evident in the literature, there does exist a broad-based *theory* of coordination (Malone and Crowston, 1994).

### 3.1. Malone and Crowston's Coordination Theory

Malone and Crowston (1994) developed an interdisciplinary theory of coordination (which we will refer to hereafter as ‘Coordination Theory’). Their Coordination Theory is based on ideas from organisation theory, economics, management, and computer science. The key idea in Coordination Theory is that coordination is needed to address dependencies, which are the *constraints on action* in a situation. Coordination is made up of one or more coordination mechanisms; each one addresses one or more dependencies in a situation. While Coordination Theory is very useful for identifying dependencies, categorising those dependencies, and identifying the coordination mechanisms in a situation, it is a ‘theory for analysis’ (Gregor, 2006, p. 623) and is not intended to be used for prediction. In particular, Coordination Theory encompasses no stated propositions or hypothesised relationships. In their 10-year retrospective of Coordination Theory research, Crowston et al. (2006) acknowledged these limitations and stated, “*challenges for future research include developing testable hypotheses (e.g. about the generality of coordination mechanisms)*” (p. 135). While Coordination Theory in its present form may not be suitable for predicting particular outcomes such as coordination effectiveness or project success, nevertheless it is a valuable tool for *better understanding* the ways in which particular activities or artefacts support coordination in organisational settings. In this study, Coordination Theory is used to analyse the dependencies and associated coordination mechanisms in agile software development projects. From that analysis a generic *coordination strategy* concept relevant to agile software development projects is developed.

Other coordination research relevant to software development projects appears in one of three contexts: organisation studies, IS projects, and teamwork. In the following sections, research in each context is presented in turn.

### 3.2. Coordination in organisation studies

Coordination is considered a key determinant of organisation structure in classic organisational theory research, and various organisation-level coordination modes have been identified (Galbraith, 1977; Malone, 1987; March and Simon, 1958; Mintzberg, 1980; Thompson, 1967; Van de Ven et al., 1976). Impersonal coordination is based upon “*a codified blueprint of action[which] is impersonally specified*” (Van de Ven et al., 1976, p. 323) and is achieved with the use of “*pre-established plans, schedules, forecasts, formalized rules, policies and procedures, and standardized information and communication systems*” (Van de Ven et al., 1976, p. 323). Minimal verbal communication is a characteristic of this mode. Coordination by mutual adjustment (Thompson, 1967) occurs at the level of either a group or an individual (Van de Ven et al., 1976). Scheduled or unscheduled meetings achieve group coordination. Personal coordination can be either vertical or horizontal. Vertical personal coordination involves communication via supervisors and line managers whereas horizontal coordination occurs via one-to-one communication between individuals in a non-hierarchical relationship.

In addition to defining various coordination modes, Van de Ven et al. (1976) identified task uncertainty, task interdependence, and size of the work unit as fundamental determinants of coordination mode. They defined four workflow categories: independent, sequential, reciprocal, and team. Each category evidences a different level of interdependence and an associated coordination mode. In particular, team mode is shown to have the highest level of workflow interdependence, and coordination takes place in group mode using scheduled and unscheduled meetings.

Considering agile methods and the agile manifesto (Beck et al., 2001) in the light of coordination theory from organisation studies, agile approaches are seen to achieve coordination by mutual adjustment at the group level (formal and informal meetings), and by personal horizontal coordination (Van de Ven et al., 1976) at the individual level. For example, in the Scrum method (Schwaber and Beedle, 2002), group mode coordination is achieved with sprint planning meetings, daily scrum meetings, and sprint review meetings. In Extreme Programming (Beck, 2000), personal horizontal coordination is achieved using pair programming and co-location. In addition, Van de Ven et al.'s (1976) team workflow concept, despite having been proposed long before agile methods were first used, has similarities with the self-organising team approach recommended in the agile manifesto (Beck et al., 2001), inasmuch as team workflow involves a group working concurrently to diagnose, collaborate, and problem-solve to process the work. Organisation studies also suggest that an agile approach may engender high coordination cost due to its reliance on mutual adjustment and group mechanisms (Van de Ven et al., 1976).

Two issues arise when applying organisation theory to agile software development or any other type of IS project. First, organisation-level coordination research is based on theoretical and empirical work carried out in the pre-1990 era and the IS organisational environment of 2011 may not be comparable. Second, using organisation theories to explain and predict agile approaches assumes an agile development project is a form of time-bounded organisation. While some researchers have made this assumption – for example, Barki et al. (2001), Nidumolu (1995), and Xia and Lee (2005) – such an assumption is questionable. Organisation level theories may not hold at the project level due to the effect of time constraints that may not be present in permanent organisations (Barki et al., 2001).

### 3.3. Coordination in information systems projects

A number of researchers have studied the role of coordination at the level of the information systems (IS) project. Nidumolu (1995) investigated the impact of project uncertainty and vertical and horizontal coordination mechanisms on project performance in 64 IT projects, and found that horizontal coordination had a positive effect on project performance. Andres and Zmud (2001) studied the impact of task interdependence, coordination strategy, and goal conflict on software development success using an experimental approach, and found that ‘organic’ coordination was generally superior in supporting success under conditions of high interdependence between organisational subunits.

McChesney and Gallagher (2004) conducted an interpretive interview-based study of coordination practices in software engineering projects. They found that in order to explain all of the coordination mechanisms in the two projects they studied, they needed to use Coordination Theory (Malone and Crowston, 1994) alongside collective mind theory (Weick and Roberts, 1993). They determined that coordination has both explicit as well as implicit components, and proposed that both types are necessary for a project to be successful.

Research into the impact of coordination on project success within IS development is diverse. A variety of contexts have



been studied (e.g. IT projects, software projects, student projects, software teams) and a variety of ‘success’ constructs have been used (e.g. project satisfaction, productivity, process success, team effectiveness) (Aladwani, 2002; Andres and Zmud, 2001; Chen et al., 2009; Kraut and Streeter, 1995; MacCormack et al., 2001; Nidumolu, 1995; Nidumolu and Subramani, 2004). Overall, this body of research has shown that effective coordination is necessary, but not sufficient, for the success of software projects (Curtis et al., 1988; Espinosa et al., 2004; Kraut and Streeter, 1995; Nidumolu, 1995).

#### 3.4. Coordination in teamwork studies

Teamwork is the third context in which coordination is a focus. Teamwork research has identified an implicit form of coordination that goes by many different names. ‘Shared cognition’ is implicit coordination that occurs within work groups without explicit speech or message passing (Rico et al., 2008). Implicit coordination can occur via shared mental models (Kang et al., 2006; Yuan et al., 2007), collective mind (Weick and Roberts, 1993), expertise coordination (Faraj and Sproull, 2000), or team mental models (Mohammed et al., 2010). Furthermore, research has linked team cognition directly with team coordination (Fiore and Salas, 2004). Although these concepts are all slightly different, taken together they suggest that there is more to coordination than the explicit forms referred to in Coordination Theory (Malone and Crowston, 1994) and in organisation theory more generally.

#### 3.5. Coordination in agile software development

Coordination is recognised as an important aspect of agile software development in a small number of theoretical and empirical studies. Classic organisation theory was used by Cao and Ramesh (2007) to examine whether the coordination mechanisms suggested by agile methods (e.g. co-located customers, short iterations) are consistent with the coordination mechanisms proposed by Van de Ven et al. (1976) (i.e. coordination by impersonal, personal, and group modes). They proposed consistencies between the classic coordination modes and agile practices in small projects but inconsistencies when using these methods in large projects. Thompson’s (1967) ideas on interdependencies and coordination (i.e. coordination by standardisation, planning, and mutual adjustment) were used by Barlow et al. (2011) to justify a theory-based methodology selection framework to guide large organisations in selecting between pure agile, pure plan-driven (Boehm and Turner, 2004) or hybrid methodologies.

Empirical research into coordination in agile software projects has identified individual practices that contribute to coordination. Pikkarainen et al. (2008) used Coordination Theory (Malone and Crowston, 1994) to study two small co-located agile projects and found sprint planning meetings, open office space, and daily meetings provide efficient communication. Used together, these practices were found to promote informal communication and substituted for documentation as a communication mechanism. Moe et al. (2010) studied a single co-located Scrum project where coordination suffered due to misapplication of Scrum practices partially caused by an existing organisational structure that promoted specialisation of skills within individuals. This ultimately resulted in team members “not knowing what the others were doing” (Moe et al., 2010, p. 488).

Product backlog, sprint backlog, scrum board, and daily meetings were practices identified for achieving coordination in a globally distributed Scrum project involving teams in India and Denmark. In this study, Pries-Heje and Pries-Heje (2011) identified coordination as one of four critical elements that explain why Scrum ‘works’ as a project management approach.

Ethnographic studies of co-located XP teams by Sharp and Robinson (2004), and MacKenzie and Monk (2004) also drew attention to specific coordination practices and artefacts. These studies identified story cards used for recording requirements, wallboards displaying story cards and their progress to completion, unit tests, card games, and code repositories as practices for achieving coordination. Hoda et al. (2010) identified a ‘coordinator role’ in their grounded theory study and found people taking that role act as conduits for the transfer of information between the agile project team and the customer group.

In summary, the various approaches for studying coordination in the extant literature all evidence shortcomings in explaining or predicting the effectiveness of agile software development. Coordination Theory is not suitable for prediction, although it is useful for identifying dependencies and coordination mechanisms. Furthermore, the three coordination research domains relevant to the study of agile software development – organisational, IS project, and teamwork – provide many coordination concepts that could be applied to the context of agile software development, and some studies have made this link. However, those studies tend to describe *which* practices act to achieve coordination, but do not attempt to develop theory using coordination as a central lens.

## 4. An initial conceptual framework

The general coordination framework shown in Fig. 1 is useful for positioning the present research. This framework is an extension and simplification of one previously proposed by Espinosa et al. (2004). Those authors melded previous coordination research, including items mentioned in the preceding literature review, into a single framework. Their framework links dependencies, implicit and explicit coordination mechanisms, and a coordination outcome concept, with team performance in an input–process–output model of coordination. Their framework has two focal coordination concepts: “coordinating” and “state of coordination.” *Coordinating* is the managing of dependencies; it is the combination of different coordination mechanisms used within a project to achieve coordination. In contrast, the *state of coordination* is the extent to which that coordination is effective. To clarify the distinction between the two concepts they are hereafter referred to as *coordination strategy*, which consists of coordination mechanisms occurring in agile software development projects, and *coordination effectiveness*.

Complexity and uncertainty were incorporated into the conceptual framework because these concepts are associated with coordination in the organisation theory literature (Van de Ven et al., 1976), in the IS project literature (Barki et al., 1993; Xia and Lee, 2005), and in the general project literature (Strauss, 1988).

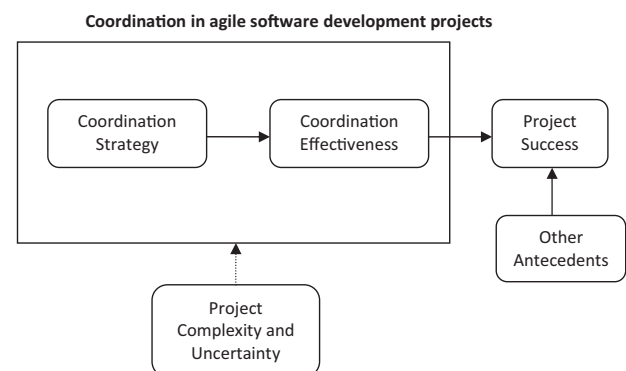


Fig. 1. Initial conceptual framework.

Source: Adapted from Espinosa et al. (2004).

**Table 1**  
Organisation, project, and data collection summary.

	Cases		
	Land	Storm	Silver
Organisation type	Government	Commercial service provider	Commercial software development firm
Organisation size	2000 in NZ	200 in Australasia, Asia and Europe	20 in NZ
Project purpose	To improve the organisations interactions with the public	To migrate a critical legacy system to a modern technology platform	To provide a new reporting system for an external client
Contractual basis	In-house development	Independent contractors working on the client site	Development for external client
Development methodology	Scrum	Scrum and XP	Scrum
Team size	6	10	5
Interviews	2	5	4
Roles of interviewees	1 Project manager 1 Software developer	1 Project manager 2 Software developers 1 Tester 1 Domain expert	1 Development manager 1 Scrum coach 2 Developers

As depicted in Fig. 1, coordination strategy, which is comprised of a combination of coordination mechanisms in a situation, determines coordination effectiveness. Coordination effectiveness is represented as an antecedent to project success, but coordination effectiveness is only one of a number of factors contributing to project success, as the literature review showed. A project may be well coordinated yet be unsuccessful for reasons unrelated to coordination, such as misinterpretation of requirements, or budgetary and resource constraints. Therefore, to maintain a focus on coordination, project success, and other antecedents are not considered further in this study. Finally, complexity and uncertainty are hypothesised to influence the project, but the exact nature of that relationship is unclear.

This coordination framework, while general, provides a useful starting point for the present research. Data gathered from a field study (discussed below) will be analysed in order to develop a more specific model of coordination which pertains directly to agile software projects.

## 5. Research method

A positivist multi-case study research approach was selected to explore the concepts of coordination strategy and coordination effectiveness based on empirical data from ongoing software development projects. This approach is a well-accepted way to investigate phenomena in natural information system development contexts where events cannot be controlled and where it is important to capture the detail in a situation (Darke et al., 1998; Eisenhardt, 1989; Pare, 2004; Yin, 2003). Furthermore, this method is suitable for building theory, which was the principal objective of the research (Eisenhardt and Graebner, 2007). To ensure accepted standards for validity and reliability, the study followed guidelines developed by Dube and Pare (2003) for carrying out rigorous exploratory positivist case study research in the field of information systems.

### 5.1. Case selection strategy

The unit of analysis – the individual case – was a software development project with co-located participants. Case selection followed a replication logic strategy, a tactic recommended in case study design (Eisenhardt, 1989; Miles and Huberman, 1994; Yin, 2003) which involves selecting cases that are similar and therefore likely to provide similar results (are literal replications) or selecting cases that are dissimilar and therefore likely to provide contrasting results for predictable reasons (are theoretical replications).

The study consisted of three literal replications. Each project was a 'typical' case selected because it was expected to show normal

or average characteristics (Pare, 2004). Case sites used the common agile method Scrum, or Scrum with some additional practices from Extreme Programming, with a team size between 2 and 10. Ten is the maximum team size recommended for effective use of these methods (Beck, 2000; Schwaber and Beedle, 2002). To distinguish this research from that addressing distributed agile software projects, all projects were required to have co-located teams. Variability across the cases was maximised by selecting 'polar types' as recommended by Eisenhardt and Graebner (2007) to provide better concept definition by accounting for a variety of contexts and factors in the emergent theoretical concepts. To ensure variability, projects were selected from different types of organisation using different types of development contracts. Key details concerning the three projects are provided in Table 1.

Certain practical considerations also influenced case selection. Wellington, the capital city of New Zealand (NZ) was the location for all selected projects. Wellington is a centre for software development because government departments and company head offices are located there. Selected projects had to be at least one-half completed or completed within one month prior to the interviews. Consequently, all of the projects studied were well underway and participants were responding to current events. The software under development had to be at least moderately complex, and be of at least moderate importance to the client or host organisation. A key informant was consulted and asked for their assessment of project uncertainty and complexity while the project was being considered for selection.

Cases were located through a not-for-profit group, the Agile Professionals Network (APN<sup>1</sup>), and from the researchers' extensive professional network. The aim of APN is to inform practitioners and their organisations about the agile approach by hosting monthly seminars on agile adoption in organisations. Contacts within APN and speakers at seminars were approached and asked to suggest potential projects for the study. A meeting was held with a key organisational contact to determine whether their project met the selection criteria. Not all people approached agreed to participate, and initial contacts were discontinued if their project did not meet the selection criteria. Three projects were eventually selected and code-named, Land, Storm, and Silver.

### 5.2. Data collection

The primary data collection method was semi-structured interviews of project team members. An interview schedule appropriate for collecting data on coordination in software

<sup>1</sup> APN: <http://www.agileprofessionals.net/>.

development projects was developed based on suggestions by Crowston (1991) for identifying coordination mechanisms and dependencies, McChesney and Gallagher (2004) for software project profiles, and Spradley (1979) for interviewing in natural situations. Spradley, for instance, suggests avoiding ‘why’ questions to ensure that participants give a full and open range of responses. Therefore questions related to coordination and dependencies were not asked directly in the form of ‘why’ questions, e.g. “Why do you have a wallboard to show tasks?”, but more indirectly as ‘how’ questions such as “how do you know what task to perform when the current task is complete?”

In each project, the first interview was conducted with one senior team member who could provide details on the project, its purpose, background, history, and issues. This person was often a project manager, but not in all cases, as not all teams included a formal project manager role. Following this interview, and depending on team size, up to four other project team members including developers, analysts, domain experts, and testers, were interviewed for 1 to 1½ hours. As well as interview transcripts, source data included field notes taken during observations made while visiting work sites and attending meetings, project documents, photographs taken at the work sites, and questionnaire data. Questionnaires were used in particular to gather data on the organisation and project details and the agile method practices used on the project. The questionnaire items were adapted from items developed for this purpose by Strode (2005).

Interviews of project team members focused on three topics: specific questions about the participants’ background, open-ended questions about the participants’ daily work practices, and each interview concluded with three specific questions designed to elicit perceptions of project coordination effectiveness:

1. What makes this project well-coordinated?
2. What interferes with coordination on this project?
3. In your opinion, based on all of the software development projects you have worked on, what is a well-coordinated project?

### 5.3. Data analysis

The first step in data analysis was to prepare a full description of each case using a common framework. This description included details of the organisation, the project, the technologies, the team, the development method, and any coordination problems in the project that emerged during the interviews. The description was sent to one project participant for verification, and any factual errors found in the description were then corrected.

Within-cases analysis was the second step in data analysis. For each case, this involved transcribing all interviews using a common template and setting up a database of data obtained from other sources. Transcript data was entered into the qualitative data analysis tool NVivo™ for ease of qualitative coding. A general inductive coding approach was followed (Miles and Huberman, 1994; Thomas, 2006) beginning with starter codes based on concepts in the initial conceptual model (i.e. coordination strategy – formed from a variety of coordination mechanisms, and project complexity and uncertainty). The aim of this analysis was to identify dependencies and their associated coordination mechanisms because a coordination mechanism is only legitimate if it addresses a dependency (Malone and Crowston, 1994). Heuristics provided by Crowston and Osborn (2003) for analysing dependencies and coordination mechanisms in a situation were followed. Their procedure is as follows:

*Dependency-focused analysis.* Identify dependencies, and then search for coordination mechanisms. In other words, look for

dependencies and then ask which activities manage those dependencies. Failure to find such activities might suggest potentially problematic unmanaged dependencies.

*Activity-focused analysis.* Identify coordination mechanisms, and then search for dependencies. In other words, identify activities in the process that appear to be coordination activities, and then ask what dependencies those activities manage. This approach asks directly whether all observed coordination activities are necessary (Crowston and Osborn, 2003, p. 352).

Each case was analysed using both dependency-focused and activity-focused analysis. The unit of data was a sentence or short paragraph, depending on the amount of conversational text required to convey the participants’ meaning.

Codes identified in the first case were used as starter codes for the second case, and codes from the second case acted as starter codes for the third case. In addition, new codes were added as they emerged from each case during the dependency and coordination mechanism analysis. By way of illustration, Table 2 lists a number of the coordination mechanism codes identified in one case, project Storm. For descriptions of the agile practices used in the projects in this study, many of which were identified as coordination mechanisms, see Strode (2005) or Williams (2010).

In Table 2, Column A shows individual coordination mechanisms first identified in the case, while Columns B and C show higher-level categories formed after identifying similarities in the coordination mechanisms in Column A. These higher-level categories formed the basis of a coordination strategy for the project. Information to support the coordination strategy concept was also drawn from the project coordination issues found in the case, and any unmanaged dependencies identified. Further, evidence of the influence of project complexity, and project uncertainty on coordination strategy was noted, along with any other antecedents influencing the coordination strategy. Outcomes of the coordination strategy were also identified to assist in forming the coordination effectiveness concept.

**Table 2**  
A sample of codes from case Storm.

A Coordination mechanism code	B Coordination mechanism category	C Frequency of occurrence
Domain specialist on team	Synchronisation activities	Project
Breakdown session		Iteration
Assignment of task to self		Iteration
Daily stand-up		Daily
Cross-team talk		Ad hoc
Continuous build		Ad hoc
Product backlog	Synchronisation artefacts	Not applicable (n/a)
Wallboard		n/a
User story		n/a
Done list		n/a
Working software		n/a
Project wiki		n/a
Workshop to generate backlog	Boundary spanning activity	Project
Software demo to user		Iteration
Dedicated time for consultation		Daily
Formal meeting		Ad hoc
Informal negotiation		Ad hoc
List of user acceptance tests	Boundary spanning artefact	Ad hoc
Single priority team		Availability
Team member co-location		Proximity
Redundant skill		Substitutability
Tester		Coordinator role
Project manager		n/a

Cross-case analysis, the third step in data analysis, involved blending the individual project coordination strategies to form a single generic coordination strategy concept. In a separate cross-case analysis, the coordination effectiveness concept was developed based on responses to the three specific questions on coordination effectiveness presented above.

## 6. Case descriptions

This section describes each of the three projects including their coordination problems, and development methodology.

### 6.1. Land

Project Land took place in a Government department. This project was highly important to the organisation and its purpose was to improve the organisation's interaction with the public by providing a better level of service on the public web site. This involved redeveloping the site using current technologies. Another aim of the project was to streamline internal processing of requests for service from the public. The IT section had experienced a lack of success with recent projects, so when this project began the project manager convinced management to trial the Scrum method. This person was self-taught in Scrum, whereas the single developer on the project had worked in a Scrum project previously in another company. None of the other team members had experience with agile development. The end-users of the software were located throughout New Zealand and had little direct influence on the project. The project had a strict time limit because the single developer was leaving on a specific date, and his expertise was not available elsewhere in the organisation, therefore it was imperative that the project be completed before he left.

Coordination problems in Land were related to the lack of consistent availability and proximity of all team members. The key developer was not working full-time, and the other team members were not devoted full-time to the project. Although the project team consisted of six people, three of them were domain and technical experts whose role was to elaborate the system requirements. These people formed a management sub-team that was not co-located with the development sub-team. This caused time delays in clarifying requirements, and the project manager had to make additional efforts to achieve coordination with the management sub-team. The project manager explained:

"...Monday, Tuesday, Wednesday was building [the software]. Thursday morning was a get together [...]. Now it wasn't supposed to be 'show and tell' of what we had done so far, because we were supposed to be communicating with them daily, but unfortunately some of the business people were involved in [another project] that was [due] to be delivered to the executive leadership team. So poor Brian was all day in meetings and so trying to get hold of him sometimes was a little bit difficult. So, we found that the Thursday morning became a catch up to see where we were all at, formally as a group, as opposed to me talking to each person individually, which I did a lot of as well. So Thursday morning became a bit of a 'show and tell', decide what user stories were going to be in the next iteration, size those, and then do the design work for it." [Land, Project manager]

The project manager explained the impact this had on her work:

"I was just forever either picking up the phone, or if it was going to be a longer conversation, that needed to show them something, I would wander upstairs, or them if they weren't around so at least the question had been posed to them, and they would get back to me and we would get together. But it

would have been so much nicer if they had been [co-located] actually." [Project manager, Land]

The underlying cause for this coordination problem was the way the organisation was structured. A matrix organisation structure meant the management sub-team members were not assigned exclusively to a single project.

Land followed a Scrum process that began with an iteration zero for planning, and then maintained a strict weekly sprint with particular activities scheduled on certain days of the week organised to accommodate the lack of availability and proximity of all team members. Each sprint included one main meeting to report progress and carry out requirements definition. Whole-team meetings outside of these scheduled meetings were uncommon. Ad hoc meetings normally involved the project manager having discussions with individual team members. A shared document management system gave all team members access to project documents. Two main project documents were used: one showed project progress in the form of user stories and their completion status, and the other was a requirements and design specification document. A wallboard to display user stories and project progress was not used in this project.

### 6.2. Storm

Project Storm took place in a quasi-public sector organisation (an SOE<sup>2</sup>). This organisation provided critical infrastructure to the New Zealand economy and worked closely with international agencies. The project involved the migration of a complex and key legacy system, which underpinned the activities of the whole organisation, to a new technology platform. When the project began, it was contracted to a project manager who selected a team of 10 contractors to develop the new system. Team members were selected because they possessed strong technical ability, had good communication skills, and were adept at working in a team. The team members all worked full-time on site in a single large room. The end-users of the software were highly skilled engineers located in the same building, in a separate room. The engineers were freely available for consultation on requirements during most working hours. However, they worked shift work, so it was not always possible to consult with a particular individual. This coordination problem was handled by forming a special group, as the tester explained:

"Well because they are on crazy shifts you know they run 24-7 at [the organisation], so they are on really wild shifts. We've managed to grab a team of four who are what we call our beta testers so generally there will be one of those persons on every day that we can always have an ear with." [Tester, Storm]

Coordination problems in Storm occurred when interacting with the wider organisation, other organisations, and the engineers. The project manager and the tester who took coordination roles within the project team primarily handled these problems. The tester was a person highly skilled at communicating both within the team and with external parties. Coordinating with other business units and understanding their needs was achieved by drafting a permanent employee of the host organisation into the team to provide additional technical domain knowledge. This person had worked for the organisation for many years, and provided an important

<sup>2</sup> State-Owned-Enterprises: "SOEs are businesses (typically companies) listed in the First Schedule to the State-Owned Enterprises Act 1986. SOEs operate as commercial businesses but are owned by the State. They have boards of directors, appointed by shareholding Ministers to take full responsibility for running the business". Sourced from The Treasury, New Zealand Government. 13 April 2011. <http://www.treasury.govt.nz/glossary/soe>.



bridge from the project to other business units within the host organisation because he knew who to go to within the organisation for information and assistance. Occasionally, hold-ups occurred at random when there was critical work occurring in the engineers group and that work always took priority over consultation with the development team.

Storm followed a flexible Scrum process. An iteration zero was used to set up the project and write an initial set of user stories. This was followed by two-week iterations, but as the project progressed, that was adjusted to a one-week timeframe. The project manager explained:

“So we started with very very structured two week iterations. What we found was when we then took all of the stories from the backlogs that we were working on for two weeks worth of work and tried to do task breakdown on them, we ended up with too much time to do a task breakdown for that much work. If you can imagine nine developers, you can get through a lot. So that would become hours and hours to get through that task breakdown and it was a pain, so we actually shifted to a bit of an informal one-week process. One week iteration, so we tend to take the next stories from the backlog and break them into tasks on a one week rotation.” [Project manager, Storm]

Otherwise, project Storm used the typical Scrum practices of posting user stories and tasks on a wallboard, having daily stand-up meetings, adjusting the wallboard tasks at these stand-ups, and having regular sessions for prioritising and sizing user stories and performing task breakdown (breaking user stories into individual tasks). Sub-teams were assigned to various sub-projects, and they had separate meetings to discuss issues and develop new stories and tasks as and when necessary. Whole-team meetings held at the start of the project, during each iteration, and daily were interspersed with ad hoc meetings between team members, and also between team members and the engineers. The tester described these ad hoc sessions thusly:

“So there might be a couple of chit chat sessions where they whiteboard some data models up, and things, and then they’ll go off and do it. Then, at the end of that they’ll probably grab a few people and say, “look, I’ll show you what I’ve done and the method’s here to call this”, and everyone’s like “great great now I know how to do it.” [Tester, Storm]

### 6.3. Silver

Project Silver took place in a small, privately owned software development company specialising in software products for Government and quasi-government agencies. The client in project Silver was a not-for-profit organisation providing services to the Health sector. The new software was to replace an outdated and inadequate records management system. The five developers were all new to agile methods when the project began, and a consultant Scrum coach was employed to assist them for most of the project.

Coordination problems in Silver were primarily due to a serious issue with inadequate responsiveness from the client and the team’s inability to consult freely with the end-users. The client organisation used management representatives to provide all requirements, but these people were not end-users of the existing system. The client did not allow the development team to interact directly with the end-user group for budgetary reasons, which created a major blockage in the progress of development. End-users could not collaborate on requirements details, the client organisation was tardy when installing a test version of the new system, and feedback in the form of acceptance testing was slow and inconsistent. Each participant noted this problem with the client and its impact on the project. Here are two perspectives on the problem:

“We definitely had co-ordination problems between us and the end user. I get the impression a lot of that was political within the [client] organisation themselves and I, my personal opinion is that was detrimental to the project it really was. The bits that worked, worked well, but yeah, there was this big bottleneck.” [Silver, Senior Developer]

“Ok, ‘what do you want to have in this dropdown of educational institutions?’ We had sent data in an Excel spreadsheet to [the client] a week ago, ‘Can you please take out those that are not used’. I had, in my calendar, every two days, follow up Neil [the client], follow up with Neil, have you had a chance to have a look at this yet, have you had a chance to look at this yet, have you found out yet, and there would be notes, nag Neil, nag Neil, nag Neil.” [Silver, Development Manager]

This problem with the client blocked progress at times and made the team change the way they dealt with tasks:

“...we had one of the columns [on the wall board], we had – apart from ‘in progress’ and ‘test complete’ – was a column called ‘blocked’. And if you found yourself blocked you put [the task sticky-note] up there, it may be you were waiting for a question from the [client] you know, we’ve asked the question we are waiting for an answer, we can’t do any more on that until we get the answer to work out what we are going to do.” [Silver, Senior Developer]

Occasionally, this situation meant the project team dropped incomplete stories from the sprint:

“we have sometimes, once or twice, taken stories out of the sprint because we couldn’t get the information. And sometimes it was relatively high. . . priority stories that we could not implement in a subsequent Sprint, because we did not have enough information because they did not come back to us.” [Scrum Coach, Silver]

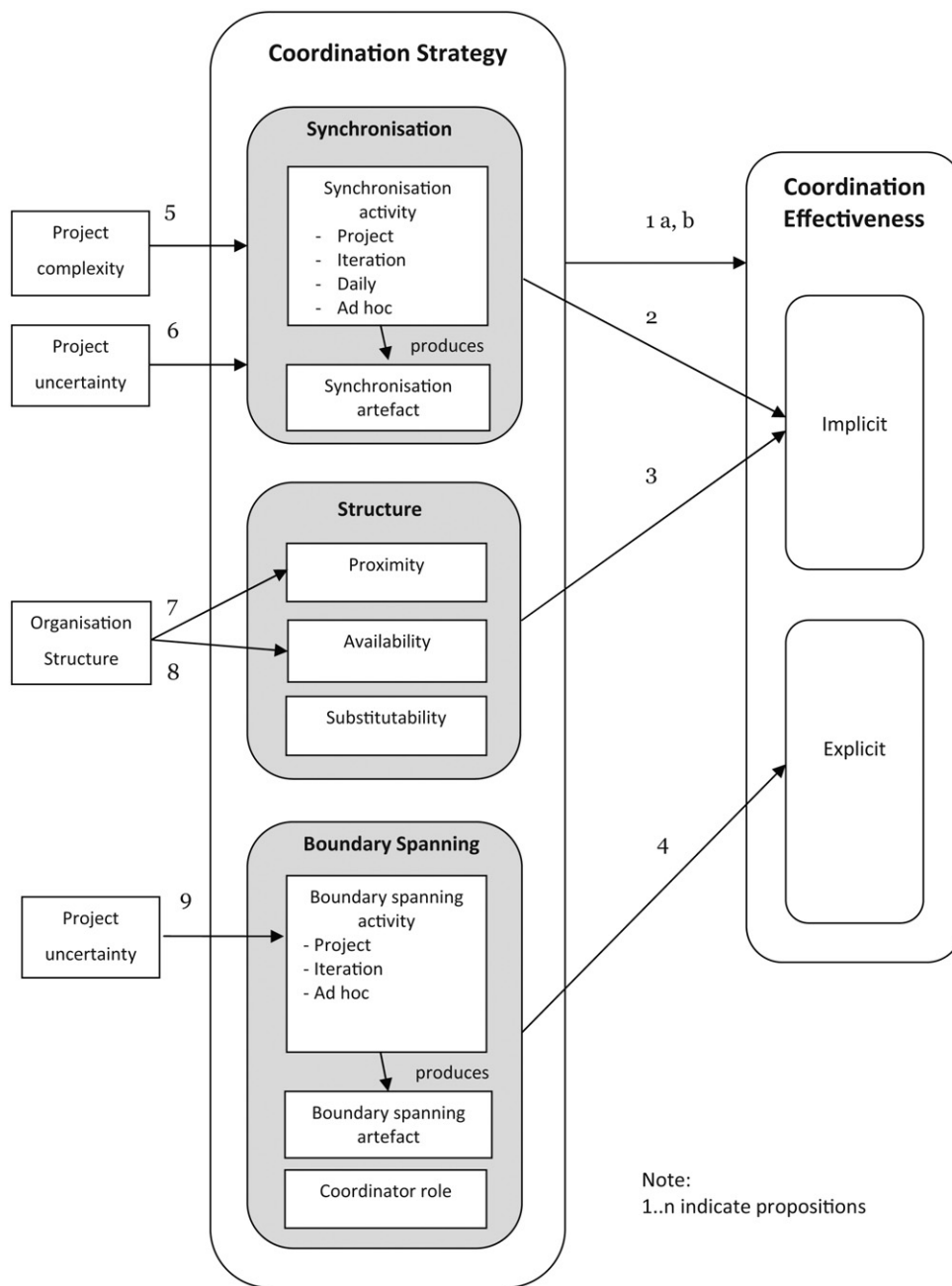
Silver followed a Scrum process with some standard XP practices. The project had an initial iteration for planning, generating ‘epics’ (high-level requirements that can be broken down into user stories), deciding on the development environment, and spiking (coding a problem to see if it is achievable). Iterations were two weeks in length throughout the project duration. The iterations involved a sprint planning meeting, a retrospective at the end of the sprint, and a product demonstration to the client. The software was released at the end of each sprint to a test environment at the client site. Daily stand-ups were used, along with numerous ad hoc meetings. The developers were all seated in the same room and easily visible to one another; as well, they could easily view the wall displaying user stories, tasks, the done list and burn down chart. The team communicated directly with one another on problems, for example when asked how they solved problems one participant said, “Shout. Just verbal. If it was something that was of a more persistent nature we’d circulate an email”. [Silver, Senior Developer]

In developing the theoretical model presented in the next section we draw on these variations in methodology use, the coordination issues encountered, and a detailed analysis of the coordination mechanisms identified in each of the three projects.

## 7. A theoretical model of coordination in co-located agile software development projects

A theoretical model of coordination in co-located agile software development projects was developed by combining evidence from each of the three cases, as described in the data analysis section. This section presents the resultant theory in three parts: coordination strategy, coordination effectiveness, and propositions defining the relationship between them. The theory is depicted in Fig. 2.





**Fig. 2.** A theory of coordination in agile software development projects.

### 7.1. The coordination strategy concept

We define a coordination strategy as a group of coordination mechanisms that manage dependencies in a situation. These mechanisms form a strategy because they are selected consciously by project stakeholders, rather than occurring by chance. In analysis, an activity or artefact was categorised as a coordination mechanism only when it was associated with one or more distinct dependencies (this article does not discuss dependencies any further for space reasons). Development of a coordination strategy concept involved first identifying the individual coordination strategy of each case and then amalgamating all strategies to form a generic coordination strategy concept. The resultant concept has three distinct components: synchronisation, structure, and boundary spanning. Table 3 provides definitions and illustrative evidence from the case data for each of these concepts and their component parts.

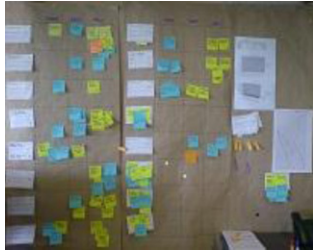
#### 7.1.1. Synchronisation

Synchronisation was achieved with synchronisation activities and synchronisation artefacts produced and used during those activities.

*Synchronisation activities* are activities that bring all of the project team members together at the same time and place for some pre-arranged purpose. These activities occur at different frequencies: once per project, once per iteration, daily, and ad hoc (i.e. as and when necessary). Activities at each of the frequencies are now considered in turn.

Project synchronisation activities occur once during the project. All projects used an initial iteration zero for discussing technical decisions, developing an initial high-level project scope in the form of epics (high-level stories), and defining initial requirements by writing user stories. This is often referred to as 'sprint zero' on Scrum projects because little production development

**Table 3**  
Definitions of coordination strategy components.

	Component	Definition	Evidence
Synchronisation	Synchronisation activity	Activities performed by all team members simultaneously that promote a common understanding of the task, process, and or expertise of other team members	<p>"I think the weekly meetings. We had a weekly kind of catch up where everybody attended and I think that was really important to just make sure everyone was on the same page and no one was drifting too much." [Land, Software Developer]</p>  <p>[Storm – photograph of wallboard]</p>
	Synchronisation artefact	An artefact generated during synchronisation activities. The nature of the artefact may be visible to the whole team at a glance or largely invisible but available. An artefact can be physical or virtual, temporary or permanent	
Structure	Proximity	This is the physical closeness of individual team members. Adjacent desks provide the highest level of proximity	<p>"We definitely had a daily team meeting on the IT side of it. The business members were not necessarily always involved in that, which is again largely around the lack of co-location. . .". [Land Project Manager]</p>
	Availability	Team members are continually present and able to respond to requests for assistance or information	<p>"But you know the developers are right there and we just turn around and involve one of them in the discussions." [Storm, Tester]</p>
	Substitutability	Team members are able to perform the work of another to maintain time schedules	<p>"... in the beginning, [Steve], who did not know C#, I had a conversation with him, about how he did not feel comfortable. So, I told him, 'I ... didn't care, I was happy to take him on. . . and, if you can't do it by yourself, get someone to help you, pair, get someone to explain it to you. I don't care. We are not going to have specialist areas where one person can only do a certain type of code'. And that went for coding, testing, and making user stories." [Scrum Coach, Silver]</p> <p>"So a Developer, if he wants his story to be done, . . . if they see a couple of test tasks hanging on there for a good couple of days, they may just say "Sam, do you want me to do these, you look pretty busy there", and they'll go off and do them." [Storm, Tester]</p>
Boundary spanning	Boundary spanning activity	Activities (team or individual) performed to elicit assistance or information from some unit or organisation external to the project	<p>"...the main people we annoy with that are the operations group, are the infrastructure. The people who look after the servers; and we suddenly go "oh by the way in two weeks we want to release this new thing to you" and they go "well I'm not sure I can be ready in two weeks" and you go [rolls his eyes]." [Storm, Project Manager]</p>
	Boundary spanning artefact	An artefact produced to enable coordination beyond the team and project boundaries. The nature of the artefact may be visible to the whole team at a glance or largely invisible but available. An artefact can be physical or virtual, temporary or permanent	<p>"...so when a [end user] does come down me it's like "I was trying out this screen Sam and it doesn't seem to work" [Storm, Tester]</p> <p>The artefact in this quote is the working software acting as a coordination mechanism between the end user and the tester</p>
	Coordinator role	A role taken by a project team member specifically to support interaction with people who are not part of the project team but who provide resources or information to the project	<p>"They [the engineers] always pop down and talk to me if they are experiencing anything, an issue, and I generally end up being a bit of an entry point for them into the rest of the team because I guess I talk to them a lot more than, you know, everyone else does. I arrange meetings." [Tester, Storm]</p>

occurs during this iteration. Project synchronisation activities also occurred *during* the project, for instance, in Storm, after a number of sprints were complete, a domain specialist from another business unit was drafted into the team to provide easy access to in-depth knowledge about the legacy system. This person was located in the project room with the team, and was continuously available for consultation by everyone on the team.

Iteration synchronisation activities occur once per iteration. An iteration (also called a 'time box' or 'sprint'), is a length of time usually consisting of a number of whole days. Land used a one-week

iteration, Storm and Silver used a two-week iteration. Within each iteration, and on all projects, a regular sequence of activities was performed. For example, in Silver, a sprint planning session was held at the start of each iteration. This involved the whole team selecting user stories from the prioritised backlog, writing manual acceptance tests for each story (each developer wrote the test for one story), decomposing stories into tasks, and finally placing the stories and tasks onto the project wallboard. Silver held a 'retrospective' at the end of the iteration when the whole team met to discuss improvement to practices, and new practices to introduce into the iteration:

“...we are very few people so [the retrospective] is a conversation, which is – what did we like, how did we perform? We set out to do velocity 45 we achieved 40. How do we feel about it? What was the reason? Is that good, is that bad? And what did we like about the last Sprint? What did we hate, did we try any of those things that we were giving two weeks and how did they go, do we want to keep them, do we want to abolish them?” [Scrum Coach, Silver]

Another example of an iteration synchronisation activity involved preparing for the product demonstration and the subsequent release of software to the client site. In Silver, this involved first selecting a demonstrator from the team using “a process of drawing straws to nominate who is going to demonstrate to the business” [Developer, Silver]. The demonstrator’s job included preparing a script for the demonstration, “a process that we could follow so that whoever was doing the demo could do it in a fairly coherent fashion.” [Developer, Silver]. This sharing of responsibility for the demonstrations extended the demonstrators knowledge of the code base because they had to come to understand story code written by other developers in the team by consulting with them directly. Test data was prepared for the demonstration and regression testing ensured the demonstration would run smoothly and the software would be suitable for release immediately following the demonstration. Sometimes the demonstrator would find bugs that had to be fixed quickly before the demonstration. The whole team would then become involved to resolve these bugs, which could arise from any stories worked on in the iteration:

“We also had the last half day, panic. Because once we started scripting the demo, it was not just testing on a user story level, but all those user stories interacting with each other, so we usually found some stuff that needed fixing. And sometimes we took out user stories, because we thought they were completed, but we found a bug, so we removed them from the demo.” [Scrum Coach, Silver]

Daily synchronisation activities occurred in all projects. This consisted of a brief whole-team meeting at the start of the day when each team member stated their current situation and any issues or problems impeding their progress. In Land, these meetings took place very informally between the project manager and the single developer who had adjacent desks. In Storm and Silver, with larger teams, a ‘stand-up’ meeting was held in front of the wallboard located on one of the walls in the room. The position of tasks on the wallboard were adjusted at these meetings to reflect their current progress to completion (see photograph in Table 3).

At 9:25 am, there was a stand-up meeting attended by all 10 project members. This was held in front of the wallboard, which took up most of one wall in the room where everyone worked. Each team member in turn described yesterday’s progress, and any issues he or she currently faced. This meeting finished at 9:35 am. [Field notes, Storm]

Ad hoc synchronisation activities occur as and when needed. In Storm, ad hoc sessions involving all of the team or sub-groups of the team were common. The people would sit together at a desk provided in a corner of their office to discuss and clarify their issues or problems. In Storm and Silver, ad hoc synchronisation was also achieved using continuous builds with automated testing which informed the whole team of the status of the software under development. Silver, the project with difficulties in getting timely feedback from their customer representatives, used a product backlog maintenance session when, at random intervals, usually about twice per sprint, the whole team would sit and discuss stories and size them. The purpose was to maintain the backlog so that it held enough stories to cover one to one-and-a-half sprints. Land, with

their team distributed around a building, reported no ad hoc synchronisation activities involving all of the team. This project relied solely on a weekly meeting and a project manager contacting team members individually during the week about any issues affecting progress such as decisions on requirements.

*Synchronisation artefacts* are things produced during synchronisation activities that contain information used by all team members in accomplishing their work. These artefacts included working software, product backlog, epics, stories, tasks, the wallboard, Done lists, burn-down charts, project information repositories such as wikis or the Rally™ application,<sup>3</sup> and shared document systems. In Land, there was no wallboard, stories were developed and stored in a shared document, and a continuously evolving design specification document was the primary source of project design information. The single developer on the project described this document as the “one source of truth” [Developer, Land] in the project.

### 7.1.2. Structure

The second component of coordination strategy is termed “structure.” We use structure in its common sense as the arrangement of, and relations between, the parts of something complex. Three categories of coordination mechanism were deemed to have structural qualities: proximity, availability, and substitutability.

*Close proximity* of the whole team is a feature of agile software development methods. All project team members should be located in the same open-plan room without divisions between desks to promote an easy flow of communication (Beck, 2000; Schwaber and Beedle, 2002). This was achieved in Storm and Silver who had special project rooms set aside for their use. In Land, close proximity was not achieved and was detrimental to coordination in the project, as discussed in the case description.

*Availability* is another feature of agile development because team members work full-time on a single project. Availability was achieved readily in Storm and Silver because project team members worked full-time on the project and were all in the room together for most of the day. In Land, lack of availability caused coordination problems. Individuals were not readily available for consultation because they were not devoted to a single project and the developer was not at work one day each week. Their mechanism for coping with this situation was to have an inflexible sequence of activities within each iteration to ensure people set aside time to participate in joint activities, for example:

“The reason for the one week iteration is our developer was part time working only four days a week so what we decided we would do was we would keep the fifth day for testing so the rest of us would test what he had built. [This also meant] we would do our iteration planning on the Thursday for the following week, while he was still here.” [Land, Project Manager]

*Substitutability* is when team members have the expertise and skills to perform the tasks of another to maintain the project schedule. Substitutability was achieved with a coordination mechanism named ‘redundant skill’, which improved coordination by reducing workflow bottlenecks. In Storm, redundant skills were apparent because the project manager selected the 10 contractors on the project team specifically for their similar skills. In Land, roles were not substitutable, although some crossover occurred by chance because the project manager had database skills, and the developer had web design skills. In Silver, the Scrum Coach made a special effort to encourage skill sharing by supporting helping behaviours (Table 3 provides a relevant quote).

<sup>3</sup> A software product for supporting the Scrum methodology.



In Storm, skill redundancy, or substitutability, improved workflow. For example, when a developer saw a test task was blocking the progress of the story he was working on, he could choose to complete the test himself, rather than wait for the tester to do the test task (Table 3 provides a relevant quote).

### 7.1.3. Boundary spanning

Boundary spanning is the third component of coordination strategy that emerged from the case analyses. Boundary spanning occurs when someone within the project must interact with other organisations, or other business units, outside of the project to achieve project goals. There are three aspects to boundary spanning: boundary spanning activities, the production of boundary spanning artefacts, and coordinator roles. Boundary spanning is included in the model because the timeliness of responses by external parties to project needs has an impact on project progress. In contrast to the synchronisation and structure components of coordination, these activities are normally not team-wide, nor are they performed in subgroups. Individuals on the team performed them when the project needed information, support, or resources from other organisational units or external organisations.

Boundary spanning activity is well recognised in studies of organisations and projects. It occurs when different organisations, or groups within organisations, separated by location, hierarchy, or function, interact to share expertise (Levina and Vaast, 2005). In the projects, boundary spanning activities occurred at three frequencies, per project, per iteration, and ad hoc. The frequency of boundary spanning activities was influenced by the way the customer, or their representatives, was situated relative to the project team. In Land, customer proxies were part of the team and consequently they participated in all of the synchronisation activities and production of synchronisation artefacts. This meant boundary spanning in Land was minimal involving occasional contact between the developer and the IT support group. In Storm, customer proxies were not part of the team and interactions with this group, the engineers, involved numerous and continuous boundary spanning activities. At project frequency, examples include an initial workshop to prioritise stories, and selection of four engineers to act as beta testers. These engineers became preferred people for team members to contact for requirements clarification and user acceptance testing feedback. Boundary spanning also occurred with the whole engineering group per iteration when the team gave software demonstrations to the engineers. Iteration frequency boundary spanning activities included sessions with the whole team and individual engineers for story creation and prioritisation. Ad hoc meetings between members of the development team and the engineers were frequent and included formal and informal meetings, and unscheduled breakdown sessions.

Another boundary spanning activity in Storm was to send a selected project team member out for extended periods to work with another business unit to learn about the requirements or technical aspects of another system that would later be integrated with the new system. Silver achieved boundary spanning with their customer proxies by adhering to a strict per iteration schedule to ensure they had regular contact and feedback from their customer. Ad hoc boundary spanning activity was severely limited due to the problems with customer availability.

Boundary spanning artefacts are physical things produced to support boundary-spanning activities and enable coordination beyond the team and project boundaries. For example, Land produced project management plans for their project management office, and Request for Change forms for the IT support unit when additional servers were required. In Storm, a single document was sent to the engineers to guide their User Acceptance Testing:

"I send them out a kind of . . . worksheet of things [a list of tests] that I would like to see them looking at, although I try not to tell them what to do, otherwise they are going to do exactly what I would have done." [Tester, Storm]

In Silver, the Scrum coach produced a project plan for the management of the organisation to provide them with feedback on project progress.

Coordinator role was another coordination mechanism to accomplish boundary spanning. In Land and Storm, the project manager acted as a coordinator between the organisation management and the project team by providing reports to management on progress, and by gathering information need by the team from other organisations and business units. In Storm, the tester acted as a coordinator between the engineers and the project team, he explained his role this way: "and I generally end being a bit of an entry point for them [the engineers] into the rest of the team because I guess I talk to them a lot more than, you know, everyone else does. I arrange meetings." [Tester, Storm].

## 7.2. The coordination effectiveness concept

"Coordination effectiveness" is the dependent variable in the model depicted in Fig. 2. A particular coordination strategy results in a certain level of coordination effectiveness. Coordination effectiveness is defined as the state of coordination achieved in a project given the execution of a particular coordination strategy; in other words, it is the *outcome* of that coordination strategy. In related work, Strode et al. (2011) describes the development of this concept and provides evidence from the same three cases presented in this article, along with one other case of non-agile software development. Their concept of coordination effectiveness is summarised briefly here.

Coordination effectiveness has an implicit and an explicit component. The explicit component encompasses the objects (persons or artefacts) involved in a project. Borrowing from Coordination Theory (Malone and Crowston, 1994) and Crowston's (2003) taxonomy of generic dependencies and coordination mechanisms, a project is coordinated effectively when a required object is in the correct place, at the correct time and in a state of readiness for use from the perspective of each individual involved in the project.

In contrast to explicit coordination, which emphasises physical objects, implicit coordination is concerned with coordination that occurs within work groups without explicit speech or message passing, as discussed in the teamwork literature on coordination. Implicit coordination encompasses five components: 'Know why', 'Know what is going on and when', 'Know what to do and when', 'Know who is doing what', and 'Know who knows what'.

'Know why' encompasses each individual working on the project understanding the overall project goal and understanding how a task contributes to that overall goal. 'Know what is going on and when' is concerned with each individual working on the project having an overall idea about the project status, that is, tasks that are currently underway and tasks that need to be performed in the future. 'Know what to do and when' is concerned with each individual working on the project knowing what task they should be working on and when they should be working on that task relative to all of the other tasks that must be completed. 'Know who is doing what' concerns each individual on the project knowing what tasks others are currently working on. Finally, 'Know who knows what' addresses expertise location. This component is supported by evidence from research conducted by Faraj and Sproull (2000) into expertise coordination in software development projects.

A definition of coordination effectiveness is as follows:

"Coordination effectiveness is a state of coordination wherein the entire agile software development team has a

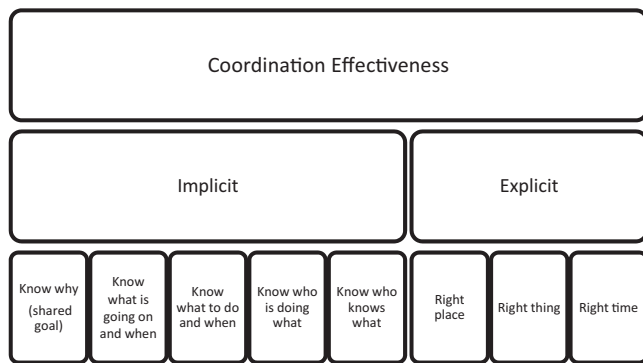


Fig. 3. Components of coordination effectiveness from Strode et al. (2011).

comprehensive understanding of the project goal, the project priorities, what is going on and when, what they as individuals need to do and when, who is doing what, and how each individual's work fits in with other team members' work. In addition, every object (thing or resource) needed to meet a project goal is in the correct place or location at the correct time and in a state of readiness for use from the perspective of each individual involved in the project." (Strode et al., 2011, p. 10)

Fig. 3 illustrates the conceptualisation of coordination effectiveness.

### 7.3. Propositions linking the coordination concepts

The coordination strategy–coordination effectiveness model shown in Fig. 2 was derived through a qualitative analysis of case study data. We propose this model not as a finished product, but as a starting point for understanding the key constructs and relationships comprising coordination in agile development projects. To elaborate the proposed model, we provide a rationale for each link in the form of tentative *propositions*. Each proposition is illustrated in Fig. 2, and discussed below.

The primary relationship proposed in this theory is that the coordination strategy employed in an agile software project leads to coordination effectiveness. However, the coordination strategy differs depending on how the customer is associated with the project.

When the customer, or their representative, is part of the team then a project coordination strategy that includes synchronisation and structural coordination mechanisms, will lead to high levels of coordination effectiveness. Synchronisation activities must occur at each frequency – project, iteration, daily, and ad hoc. Synchronisation artefacts must be produced at each frequency and the nature of the artefact must be visible to the whole team at a glance or largely invisible but available (e.g. available in Rally™). An artefact can be physical or virtual, temporary or permanent. Structural coordination mechanisms, that is, proximity, availability, and substitutability must all be at high levels. When the customer is part of the team, this attenuates coordination mechanisms for boundary spanning. These mechanisms are still necessary for interactions with external parties, for example requesting a server from the IT section, but this form of coordination is needed relatively infrequently.

However, when the customer, or their representative, is *not* part of the team then the project coordination strategy becomes more complex. In this situation, the coordination strategy includes the coordination mechanisms of synchronisation and structure *along with* multiple coordination mechanisms for boundary spanning. Boundary spanning then includes boundary-spanning activities at each of the frequencies – project, iteration, and ad hoc, and the production of boundary spanning artefacts at each of these

frequencies. In addition, it may become necessary for one or more project team members to take a coordinator role. When the customer is external to the team, boundary-spanning coordination mechanisms increase in frequency because they are the means by which a continuous supply of information is fed into the project. This information comes from customers, and is necessary for building the product backlog and for providing requirements and feature details. Information may also come via informal (ad hoc) feedback from the customer on the quality of working software, or more formally as the result of user acceptance testing. This leads to Proposition 1, which has two parts:

**Proposition 1a.** *A coordination strategy that includes synchronisation and structure coordination mechanisms improves project coordination effectiveness when the customer is included in the project team. Synchronisation activities and associated artefacts are required at all frequencies – project, iteration, daily, and ad hoc.*

**Proposition 1b.** *A coordination strategy that includes synchronisation, structure, and boundary spanning coordination mechanisms improves project coordination effectiveness when the customer is an external party to the project. Synchronisation activities and associated artefacts are required at all frequencies – project, iteration, daily, and ad hoc. Boundary spanning activities and associated artefacts are required at all frequencies – project, iteration, and ad hoc.*

Proposition 1 treats coordination effectiveness as a unitary concept and makes no distinction between the effect of coordination strategy on the implicit and explicit components of coordination effectiveness. Many coordination mechanisms contribute to both implicit and explicit coordination. However, some coordination mechanisms tend to promote implicit coordination whereas others promote explicit coordination. The following propositions reflect this tendency. Note that in the following discussion the components of implicit coordination, as defined in the coordination effectiveness section, are italicised.

Synchronisation activities and their associated artefacts increase implicit coordination. For example, the whole team is able to find out *what is going on and when*, and *knows what to do and when* by participating in epic creation during iteration zero, and by looking at the current state of the wallboard displaying currently open stories and the completion status of tasks. The team is able to *know who is doing what* by viewing the avatar attached to a task, by 'asking the room' [Silver, Developer], or by attending and participating in the daily stand-up meeting. Therefore:

**Proposition 2.** *Synchronisation activities at all frequencies – project, iteration, daily, and ad hoc, along with their associated synchronisation artefacts, increase implicit coordination effectiveness.*

When project team members are in close proximity, especially when they are in the same room with adjacent desks, they become more aware of the work of other team members by observing and over-hearing their activities. They may also become familiar with how their own task fits with others' tasks; in other words, they *know why* they are performing their task. In addition, they learn *what is going on and when*, they come to *know who is doing what*, and they become aware of *who knows what*.

When a project team member is consistently and freely available, other team members can readily consult that person whenever they need to and at short notice. Availability therefore raises project team members' awareness about *who knows what* in the project.

When project team members can perform each other's tasks because they have overlapping skill sets this is substitutability. Substitutability raises awareness about *who knows what* because to perform the work of another, you come to understand what they know and do not know. Therefore:

**Proposition 3.** *Structural coordination mechanisms i.e. close proximity, high availability, and high substitutability, increase implicit coordination effectiveness.*

The purpose of boundary spanning is to acquire resources for the project. Resources may be physical (e.g. servers) or informational (information about requirements or the technical domain). Activities such as meetings with vendors and customers, artefacts such as official requests, and someone on the project team taking a coordinator role all contribute to boundary spanning and ensuring that required resources, *the right things, are in the right place, at the right time*, so that project progress is not hindered in any way. Therefore:

**Proposition 4.** *High levels of boundary spanning coordination mechanisms, i.e. boundary spanning activities at all frequencies – project, iteration, and ad hoc, their associated boundary spanning artefacts, and a coordinator role, increases explicit coordination effectiveness.*

Project complexity can be handled by increasing the frequency of iterations. Project Storm used this tactic to cope better with unmanageably complex story-breakdown sessions. Shorter iterations mean fewer open stories and therefore fewer tasks to address in the sprint. This focused the team on a smaller subset of overall requirements and reduced the complexity of the overall task by limiting the number of factors to consider at one time. Therefore:

**Proposition 5.** *Under conditions of high project complexity, increasing the frequency of iteration and ad hoc synchronisation activities will maintain coordination effectiveness. The production of related synchronisation artefacts must be adjusted accordingly.*

Project uncertainty primarily takes the form of uncertainty in requirements, but can also include such things as uncertainty about tools, techniques, infrastructure, and other factors. Silver had high project uncertainty caused by their customer who could not provide requirements details or feedback on the quality of working software in a timely manner. To cope with this the team used a tactic of putting blocked stories and their tasks on hold, opening additional stories during a sprint to ensure some stories were completed by the end of the sprint, and story switching (de-prioritising blocked stories and opening lower priority stories). Therefore:

**Proposition 6.** *Under conditions of high project uncertainty, to maintain synchronisation activity frequency and production of associated artefacts, changing the priority of stories will maintain coordination effectiveness.*

Organisations may choose any number of ways to organise their project work. Some organisations choose a mono-project structure whereby each project has a project team who are devoted full-time to a single project. In others, a matrix or multi-project structure means project team members work simultaneously on multiple projects, or work on a project while also performing operational duties. Land, as explained in the case description, used a matrix organisation structure, and this had an impact on both team member proximity and availability. Therefore:

**Proposition 7.** *A mono-project organisation structure enables close proximity relative to multi- or matrix structures.*

**Proposition 8.** *A mono-project organisation structure improves availability relative to multi- or matrix style structures.*

In the situation where the source of requirements (that is, the customer or end-user) is isolated from the team rather than co-located and part of the team, increased uncertainty has a slightly different affect to that described in Proposition 6. When uncertainty is high and the customer is not readily available, it is possible to maintain coordination effectiveness by increasing boundary spanning coordination mechanisms. In Storm, this involved setting up beta testers

to become a primary source of requirements details, and then initiating additional ad hoc meetings with them when necessary. In addition, the tester on the team took on a coordinator role between the engineers and the project development team. Therefore:

**Proposition 9.** *Under conditions of high project uncertainty, when the customer is not part of the team, increased boundary spanning coordination mechanisms will maintain coordination effectiveness. The production of related boundary spanning artefacts must be adjusted accordingly.*

## 8. Discussion

This study addressed two research questions. The first asked how coordination is achieved when using an agile development approach. This study has shown that an agile software project coordination strategy is achieved through three distinct categories of coordination mechanism – synchronisation, structure, and boundary spanning – that together enhance implicit and explicit coordination effectiveness.

Synchronisation is achieved with a variety of synchronisation activities involving the whole team and by producing artefacts during these activities that were used later as coordination mechanisms as the project progresses (e.g. stories, tasks, test suites, and working software). These activities occur at different frequencies over the project lifecycle including one-off activities that occurred once per project (e.g. iteration zero workshops with the customer to scope the project), activities that occur at each iteration (e.g. planning game and retrospective), activities that occur daily (e.g. daily stand up meeting), and ad hoc activities (e.g. cross-team talk, unscheduled story breakdown sessions). Structural coordination mechanisms formed three categories: proximity, availability, and substitutability of project team members. Boundary spanning is achieved with activities and artefacts but does not involve the whole team, but rather individuals interacting with external parties to ensure a flow of requirements and resources needed to maintain project progress. Coordinator roles may also be necessary when stakeholder groups are large and complex and particularly when the customer is external to the team and cannot work with alongside them on a daily basis.

The second research question asked about the relationship between the coordination strategy in these projects and project coordination effectiveness. The theoretical model proposes that there is a relationship between an agile coordination strategy and project coordination effectiveness. Acting together, synchronisation, structure, and boundary spanning components affect explicit and implicit coordination. For example, when the project team meets to demonstrate the working software (a synchronisation artefact) to the client (a synchronisation activity) this involves a current functioning version of the software (the right thing is in the right place, at the right time). As another example, when the project team has a daily stand up meeting (synchronisation activity), and this leads to each team member being better informed about who is performing what task on the project on that day, this contributes to the implicit component of coordination effectiveness (i.e. know what is going on and when).

Aspects of this theoretical model may have some relevance in non-agile and more traditional software projects. However, in traditional projects certain coordination mechanisms appearing in the theoretical model would be absent. These include iteration and daily synchronisation activities and many of their associated artefacts (wallboards, tasks), and a high level of substitutability. Substitutability is unlikely in traditional projects because role specialisation is considered a more effective strategy. Further, although close proximity and high availability may occur in non-agile projects they are not generally implemented as a specific



strategy to improve coordination. Boundary spanning activities and artefacts certainly occur in non-agile projects; however, they would not occur at the same frequency, because iteration is not generally used in such projects.

This study has limitations. The qualitative data were drawn from only three cases and a relatively small number of participants, therefore the empirical grounding of the coordination model is limited and the findings necessarily tentative. The findings, and the model of coordination effectiveness are based upon the responses of co-located teams; the validity of these findings in the case of distributed teams has not been established. Another limitation is that all of the projects used Scrum and XP practices. Since many other agile methods exist, the theory derived from the case data may have less relevance for practitioners using other agile methods.

The theoretical model has limitations because interaction effects between synchronisation, structure, and boundary spanning coordination mechanisms are not considered. We have made the assumption that all of these forms of coordination are needed in an effective strategy, however some may be more important than others. Further, other coordination mechanisms may exist that we have not considered due to our small number of cases. For example, the effect of team duration and individual expertise on coordination was not identified in our study but may be involved in coordination. Further research may warrant their inclusion in the model.

### 8.1. Practical implications

Although each agile method was originally conceived as a coherent set of practices, practitioners commonly select individual practices from a method rather than implementing the full method (Conboy and Fitzgerald, 2007). It is also common to find practitioners combining practices from two or more methods (Fitzgerald et al., 2006). As a practical matter, when practitioners select practices, either from a single agile method or from multiple agile methods, they can use the model presented here as an aid in identifying which agile practices to choose to achieve coordination coverage. The model suggests that practices should be selected that provide synchronisation at all of the frequencies in the model: project, iteration, daily, and ad hoc. Artefacts can be produced at all these frequencies and contribute to coordination. In addition, practitioners should consider whether their organisational structure will ensure that proximity and availability are achievable for all team members. Substitutability should be encouraged rather than discouraged. Furthermore, boundary spanning is an issue not addressed adequately in agile methods. XP provides no guidance on this issue, although in Scrum, the person who takes the role of Scrum Master acts to solve problems hindering project progress, which could include boundary spanning activities. From a coordination perspective, this role is important for handling interactions and negotiating effective support from external parties, while freeing other project team members to focus on internal project issues.

Another finding with implications for practice is that iteration length provides a way to manage an effective level of coordination. That is, under conditions of complexity, reducing iteration length increases the frequency of synchronisation activities but maintains effective coordination. It is less clear from this study how projects can effectively manage requirements uncertainty when external parties (customers or end-users) are unwilling or unable to work closely with the team. This problem may be mitigated to some extent by maintaining formal contact with these parties at least once per iteration.

Distributed software development is common in some parts of the world and efforts to use agile software development in this context are not as simple as in the co-located situation. This model provides a starting point for understanding coordination in

co-located projects, and could be extended to the case of globally distributed software development.

Operationalisation of the coordination effectiveness construct would provide a valuable measure of coordination effectiveness in agile (and possibly other) projects. Such a measure could be used to assess coordination effectiveness at various time points during a project, providing a profile of project coordination, and an early warning signal when coordination problems begin to occur. This would help organisations understand how their projects are performing with respect to coordination over the project lifecycle. It could also assist organisations to identify and address their coordination problems in a timely fashion, and improve the likelihood of successful agile project completion.

## 9. Conclusion

Information systems development has suffered from a dearth of research exploring and explaining agile software development, in particular the role of coordination in agile projects. In this article, we bring these two streams of research together to contribute to an understanding of how such projects are organised to achieve effective coordination. In addition, coordination theory has been extended to provide evidence for the relationship between coordination strategy, what you 'do' to achieve coordination, and coordination effectiveness, your 'state of coordination'.

For practitioners undertaking or adapting their agile approach, this theoretical model provides guidance in a number of areas. When choosing agile practices, many practitioners lack guidance on how different practices ought to work together to achieve an effective coordination strategy. The theory proposed here suggests that consideration should be given to adopting synchronisation activities, including practices at project, iteration, daily, frequencies and incorporating ad hoc activities as well. In addition, the organisation structure in which the agile project occurs should be a consideration because it is an important enabler of effective coordination by supporting staff availability and proximity. Role generalisation (what we have named substitutability) also enhances coordination in agile projects, and consideration of coordination in boundary spanning is also needed, because although boundary spanning may be partially external to a project, it also has an influence on how effectively a project is coordinated.

Future work is needed to confirm the efficacy of this theoretical model using additional empirical tests. In particular, future research should aim to verify our conceptualisation of coordination strategy (i.e. certain agile practices acting as coordination mechanisms) and its effect on coordination effectiveness. The theory might also be used to compare coordination between software projects, and between different types of software projects, for example agile and non-agile projects, distributed and co-located projects, and between different system development methodologies.

This research provides a preliminary overall model depicting how the fundamental concept of coordination is achieved in agile software development projects. Agile practices and organisation level activities and artefacts that support coordination can now be investigated in light of the model, and coordination theorists may choose to use this theoretical model to gain interesting new insights into coordination in other contexts.

## Acknowledgements

We wish to acknowledge the assistance of the contributing organisations and especially the individual participants. These people gave freely of their time to share their project experiences, and we are grateful for that. We also acknowledge the assistance

of the Wellington Agile Professionals Network in locating suitable projects.

## References

- Abrahamsson, P., Conboy, K., Wang, X., 2009. 'Lots done, more to do': the current state of agile systems development research. *European Journal of Information Systems* 18, 281–284.
- Abrahamsson, P., Warsta, J., Siponen, M.K., Ronkainen, J., 2003. New directions on agile methods: a comparative analysis. In: *Proceedings of the 25th International Conference on Software Engineering, ICSE'03*, IEEE Computer Society, Washington, DC, USA, pp. 244–254.
- Agerfalk, P.J., Fitzgerald, B., Slaught, S.A., 2009. Flexible and distributed information systems development: state of the art and research challenges. *Information Systems Research* 20, 317–328.
- Aladwani, A., 2002. An integrated performance model of information systems projects. *Journal of Management Information Systems* 19, 185–210.
- Ambler, S.W., 2009. Agile Adoption Rate Survey Results: February 2008.
- Andres, H.P., Zmud, R.W., 2001. A contingency approach to software project coordination. *Journal of Management Information Systems* 18, 41–70.
- Avison, D., Fitzgerald, G., 2006. *Information Systems Development: Methodologies, Techniques and Tools*, 4th ed. McGraw-Hill, London.
- Barki, H., Rivard, S., Talbot, J., 1993. Toward an assessment of software development risk. *Journal of Management Information Systems* 10, 203–225.
- Barki, H., Rivard, S., Talbot, J., 2001. An integrative contingency model of software project risk management. *Journal of Management Information Systems* 17, 37–69.
- Barlow, J., Giboney, J., Keith, M., Wilson, D., Schuetzler, R., 2011. Overview and guidance on agile development in large organizations. *Communications of the Association for Information Systems* 29, 25–44.
- Baskerville, R., Pries-Heje, J., Ramesh, B., 2007. The enduring contradictions of new software development approaches: a response to 'persistent problems and practices in ISD'. *Information Systems Journal* 17, 241–245.
- Beck, K., 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., 2001. Manifesto for agile software development. Retrieved 26 January 2012 from <http://www.agilemanifesto.org>.
- Boehm, B., Turner, R., 2004. *Balancing Agility and Discipline*. Addison-Wesley, Boston.
- Cao, L., Ramesh, B., 2007. Agile software development: ad hoc practices or sound principles? *IT Professional*, 41–47.
- Chen, H.-G., Jiang, J.J., Klein, G., Chen, J.V., 2009. Reducing software requirement perception gaps through coordination mechanisms. *The Journal of Systems and Software* 82, 650–655.
- Conboy, K., 2009. Agility from first principles: reconstructing the concept of agility in information systems development. *Information Systems Research* 20, 329–354.
- Conboy, K., Fitzgerald, B., 2007. The views of experts on the current state of agile method tailoring. In: McMaster, T., Wastell, D., Ferneley, E., DeGross, J. (Eds.), *IFIP International Federation for Information Processing*. Springer, Boston, pp. 217–234.
- Conboy, K., Fitzgerald, B., Golden, W., 2005. Agility in information systems development: a three-tiered framework. In: Baskerville, R., Mathiassen, L., Pries-Heje, J., DeGross, J. (Eds.), *Business Agility and Information Technology Diffusion: IFIP TC8 WG 8.6 International Working Conference*, May 8–11, 2005, Atlanta, Georgia, U.S.A. Springer, New York, pp. 36–49.
- Crowston, K., 1991. *Towards a Coordination Cookbook: Recipes for Multi-agent Action*. Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 352 pp.
- Crowston, K., 2003. A taxonomy of organizational dependencies and coordination mechanisms. In: Malone, T.W., Crowston, K., Herman, G.A. (Eds.), *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, Cambridge, MA, pp. 86–108.
- Crowston, K., Osborn, C.S., 2003. A coordination theory approach to process description and redesign. In: Malone, T.W., Crowston, K., Herman, G.A. (Eds.), *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, Cambridge, MA, pp. 335–370.
- Crowston, K., Rubleske, J., Howison, J., 2006. Coordination theory: a ten-year retrospective. In: Zhang, P., Galletta, D. (Eds.), *Human-Computer Interaction and Management Information Systems: Foundations*. M.E. Sharpe, Armonk, New York, pp. 120–138.
- Curtis, B., Krasner, H., Iscoe, N., 1988. A field study of the software design process for large systems. *Communications of the ACM* 31, 1268–1287.
- Darke, P., Shanks, G., Broadbent, M., 1998. Successfully completing case study research: combining rigour, relevance and pragmatism. *Information Systems Journal* 8, 273–289.
- Dingsoyr, T., Dyba, T., Abrahamsson, P., 2008. A preliminary roadmap for empirical research on agile software development. In: *Proceedings of the Agile 2008 Conference*, pp. 83–94.
- Dingsoyr, T., Dyba, T., Moe, N.B., 2010. *Agile Software Development: Current Research and Future Directions*. Springer-Verlag, Heidelberg.
- Dube, L., Pare, G., 2003. Rigor in information systems positivist case research: current practice, trends, and recommendations. *MIS Quarterly* 27, 597–635.
- Dyba, T., Dingsoyr, T., 2008. Empirical studies of agile software development: a systematic review. *Information and Software Technology* 50, 833–859.
- Eisenhardt, K.M., 1989. Building theories from case study research. *The Academy of Management Review* 14, 532–550.
- Eisenhardt, K.M., Graebner, M.E., 2007. Theory building from cases: opportunities and challenges. *Academy of Management Journal* 50, 25–32.
- Espinosa, A.J., Lerch, F.J., Kraut, R.E., 2004. Explicit versus implicit coordination mechanisms and task dependencies: one size does not fit all. In: Salas, E., Fiore, S.M. (Eds.), *Team Cognition: Understanding the Factors that Drive Process and Performance*. American Psychological Association, Washington, DC, pp. 107–129.
- Faraj, S., Sproull, L., 2000. Coordinating expertise in software development teams. *Management Science* 46, 1554–1568.
- Fiore, S.M., Salas, E., 2004. Why we need team cognition. In: Salas, E., Fiore, S.M. (Eds.), *Team Cognition*. American Psychological Association, Washington, DC, pp. 235–248.
- Fitzgerald, B., Hartnett, G., Conboy, K., 2006. Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems* 15, 200–213.
- Fruhling, A., de Vreede, G.-J., 2006. Field experiences with eXtreme programming: developing an emergency response system. *Journal of Management Information Systems* 22, 39–68.
- Galbraith, J.R., 1977. *Organization Design*. Addison-Wesley, Reading, MA.
- Gregor, S., 2006. The nature of theory in information systems. *MIS Quarterly* 30, 611–642.
- Harris, M., Collins, R.W., Hevner, A.R., 2009. Control of flexible software development under uncertainty. *Information Systems Research* 20, 400–419.
- Hilkka, M.-R., Tuure, T., Matti, R., 2005. Is extreme programming just old wine in new bottles: a comparison of two cases. *Journal of Database Management* 16, 41–61.
- Hoda, R., Noble, J., Marshall, S., 2010. Organizing self-organizing teams. In: *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE 2010)*, Cape Town, South Africa, May 2–9, 2010.
- Iivari, J., Hirschheim, R., Klein, H.K., 2004. Towards a distinct body of knowledge for Information Systems experts: coding ISD process knowledge in two IS journals. *Information Systems Journal* 14, 313–342.
- Kang, H.-R., Yang, H.-D., Rowley, C., 2006. Factors in team effectiveness: cognitive and demographic similarities of software development team members. *Human Relations* 59, 1681–1710.
- Kautz, K., Madsen, S., Norbjerg, J., 2007. Persistent problems and practices in information systems development. *Information Systems Journal* 17, 217–239.
- Kotlarsky, J., van Fenema, P.C., Willcocks, L.P., 2008. Developing a knowledge-based perspective on coordination: the case of global software projects. *Information and Management* 45, 96–108.
- Kraut, R.E., Streeter, L.A., 1995. Coordination in software development. *Communications of the ACM* 38, 69–81.
- Levina, N., Vaast, E., 2005. The emergence of boundary spanning competence in practice: implications for implementation and use of information systems. *MIS Quarterly* 29, 335–363.
- MacCormack, A., Verganti, R., Iansiti, M., 2001. Developing products on "Internet Time": the anatomy of a flexible development process. *Management Science* 47, 133–150.
- MacKenzie, A., Monk, S., 2004. From cards to code: how extreme programming embodies programming as collective practice. *Computer Supported Cooperative Work* 13, 91–117.
- Madsen, S., 2007. Conceptualising the causes and consequences of uncertainty in IS development organisations and projects. In: Osterle, H., Schelp, J., Winter, R. (Eds.), *15th European Conference on Information Systems*. St. Gallen, Switzerland, pp. 855–864.
- Malone, T.W., 1987. Modeling coordination in organizations and markets. *Management Science* 33, 1317–1332.
- Malone, T.W., 1988. *What is Coordination Theory?* Alfred P. Sloan School of Management, Massachusetts Institute of Technology.
- Malone, T.W., Crowston, K., 1994. The interdisciplinary study of coordination. *ACM Computing Surveys* 26, 87–119.
- March, J.G., Simon, H.A., 1958. *Organization*. Wiley, New York.
- Maruping, L.M., Venkatesh, V., Agarwal, R., 2009. A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research* 20, 277–399.
- McChesney, I.R., Gallagher, S., 2004. Communication and coordination practices in software engineering projects. *Information and Software Technology* 46, 473–489.
- Miles, M.B., Huberman, A.M., 1994. *Qualitative Data Analysis*, 2nd ed. SAGE Publications Inc., Thousand Oaks.
- Mintzberg, H., 1980. Structure in 5's: a synthesis of the research on organization design. *Management Science* 26, 322–341.
- Moe, N., Dingsoyr, T., Dyba, T., 2010. A teamwork model for understanding an agile team: a case study of a Scrum project. *Information and Software Technology* 52, 480–491.
- Mohammed, S., Ferzandi, L., Hamilton, K., 2010. Metaphor no more: a 15-year review of the team mental model construct. *Journal of Management* 36, 876–910.
- Nerur, S., Balijepally, V., 2007. Theoretical reflections on agile development methodologies. *Communications of the ACM* 50, 79–83.
- Nidumolu, S., 1995. The effect of coordination and uncertainty on software project performance: residual performance risk as an intervening variable. *Information Systems Research* 6, 191–219.

- Nidumolu, S.R., Subramani, M., 2004. The matrix of control: combining process and structure approaches to managing software development. *Journal of Management Information Systems* 20, 159–196.
- Olle, T.W., Sol, H.C., Tully, C., 1983. Information systems design methodologies: a feature analysis. In: *Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies*. North-Holland, Amsterdam.
- Pare, G., 2004. Investigating information systems with positivist case study research. *Communications of the Association for Information Systems* 13, 233–264.
- Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., Still, J., 2008. The impact of agile practices on communication in software development. *Journal of Empirical Software Engineering* 13, 303–337.
- Pries-Heje, L., Pries-Heje, J., 2011. Why Scrum works. In: *Agile Conference 2011*, IEEE Computer Society.
- Rajlich, V., 2006. Changing the paradigm of software engineering. *Communications of the ACM* 49, 67–70.
- Rico, R., Sanchez-Manzanares, M., Gil, F., Gibson, C., 2008. Team implicit coordination processes: a team knowledge-based approach. *Academy of Management Review* 33, 163–184.
- Sarker, S., Munson, C.L., Sarker, S., Chakraborty, S., 2009. Assessing the relative contribution of the facets of agility to distributed systems development success: an analytic hierarchy process approach. *European Journal of Information Systems* 18, 285–299.
- Sarker, S., Sarker, S., 2009. Exploring agility in distributed information systems development teams: an interpretive study in and offshoring context. *Information Systems Research* 20, 440–461.
- Schwaber, K., Beedle, M., 2002. *Agile Software Development With Scrum*. Prentice Hall, Upper Saddle River, NJ.
- Sharp, H., Robinson, H., 2004. An ethnographic study of XP practice. *Empirical Software Engineering* 9, 353–375.
- Spradley, J.P., 1979. *The Ethnographic Interview*. Holt, Rinehart and Winston, New York.
- Strauss, A., 1988. The articulation of project work: an organization process. *The Sociological Quarterly* 29, 163–178.
- Strode, D.E., 2005. *The Agile Methods: An Analytical Comparison of Five Agile Methods and an Investigation of Their Target Environment*. Department of Information Systems, Massey University, Palmerston North, New Zealand, Retrieved 26 January 2012 from <http://hdl.handle.net/10179/515>.
- Strode, D.E., Hope, B., Huff, S.L., Link, S., 2011. Coordination effectiveness in an agile software development context. In: *Pacific Asia Conference on Information Systems, PACIS 2011*, Brisbane, Australia.
- Thomas, D.R., 2006. A general inductive approach for analyzing qualitative evaluation data. *American Journal of Evaluation* 27, 237–246.
- Thompson, J.D., 1967. *Organization in Action*. McGraw-Hill, Chicago.
- Van de Ven, A.H., Delbecq, A.L., Koenig, R., 1976. Determinants of coordination modes within organizations. *American Sociological Review* 41, 322–338.
- Weick, K.E., Roberts, K.H., 1993. Collective mind in organizations: heedful interrelating on flight decks. *Administrative Science Quarterly* 38, 357–381.
- West, D., Grant, T., 2010. Agile Development: Mainstream Adoption has Changed Agility. Forrester Research Inc, pp. 1–20.
- Williams, L., 2010. Agile software development methodologies and practices. In: Zelkowitz, M.V. (Ed.), *Advances in Computers*. Elsevier, Amsterdam, pp. 1–44.
- Xia, W., Lee, G., 2005. Complexity of information systems development projects: conceptualization and measurement development. *Journal of Management Information Systems* 22, 45–83.
- Yin, R.K., 2003. *Case Study Research*, 3rd ed. Sage Publications, Thousand Oaks.
- Yuan, M., Vogel, D., Zhang, X., Chen, Z., Chu, X., 2007. Antecedents of coordination effectiveness of software developer dyads from interacting teams: an empirical investigation. In: *11th Pacific-Asia Conference on Information Systems*.

**Diane E. Strode** is a PhD candidate in Information Systems and Contract Lecturer at Victoria University of Wellington, School of Information Management, New Zealand. Currently she investigates agile software development from a coordination perspective with the aim of building theory in this complex area. Her research interests include system and software development methodology and factors influencing software project success. She has extensive teaching experience in the Polytechnic and University sectors covering analysis and design, software development, and project management. She also has software development experience in an international company. She has a MInfSc (Information Systems) from Massey University, New Zealand.

**Sid L. Huff** is Professor of Information Systems at Victoria University of Wellington, New Zealand. His teaching and research address IS strategy, IT governance and IS management. His work has appeared in numerous academic and practitioner journals, including *MIS Quarterly*, *Information Systems Research*, *Journal of MIS*, *Journal of Strategic Information Systems*, and *Communications of the ACM*. His most recent book is *Managing IT Professionals in the Internet Age*, co-authored with Dr. Pak Yoong. He currently serves as Senior Editor for *Information Systems Management Journal* and for *Journal of Information Technology*, and as Associate Editor for *MIS Quarterly*. He has also written over 60 teaching cases for educational use. He received his PhD in Information Systems from the M.I.T. Sloan School of Management.

**Beverley Hope** is employed in the School of Information Management at Victoria University of Wellington teaching in areas that focus on management of Information Systems. She also serves on the School's research degrees committee. Her research interests include service quality in online environments, knowledge management, and information systems education. She has acted as editor or reviewer for many journals and conferences, and held responsibility as Conference Chair and Program Director for regional and international conferences. Beverley holds a PhD in Communication and Information Science from the University of Hawaii and an MBA from the University of Kansas.

**Sebastian Link** received a PhD in Information Systems from Massey University (New Zealand) in 2005. Currently, he is an associate professor at the School of Information Management, the Victoria University of Wellington (New Zealand). His research interests include conceptual modeling, data management, e-commerce and software verification. Sebastian has published more than 75 research papers, and served as a reviewer for numerous conferences and journals. He is a member of the editorial board of the journal *Information Systems*.