A THEORY OF COORDINATION IN
AGILE SOFTWARE DEVELOPMENT PROJECTS

BY

DIANE ELIZABETH STRODE

A thesis

submitted to the Victoria University of Wellington

in fulfilment of the requirements for the degree of

Doctor of Philosophy

Victoria University of Wellington

2012

ABSTRACT

Agile software development offers a deceptively simple means to organise complex multi-participant software development while achieving fast delivery of quality software, meeting customer requirements, and coping effectively with project change. There is little understanding, however, of how agile software development projects achieve effective coordination, a critical factor in successful software projects.

Agile software development provides a unique set of practices for organising the work of software projects, and these practices seem to achieve effective project coordination. Therefore, this thesis takes a coordination perspective to explore how agile software projects work, and why they are effective. The outcome of this research is a theory of coordination in co-located agile software development projects.

To build a coordination theory, evidence was drawn from a multi-case study following the positivist tradition in information systems. Three cases of agile software development contributed to the theory, along with one additional non-agile project that contributed contrasting evidence.

The findings show that agile software development practices form a coordination strategy addressing three broad categories of dependency: knowledge dependencies, task dependencies, and resource dependencies. Most coordination is for managing requirement, expertise, historical, and task allocation dependencies; all forms of knowledge dependency. Also present are task dependencies, which include activity or business process dependencies, and resource dependencies, which include technical or entity dependencies.

The theory of coordination explains that an agile coordination strategy consists of coordination mechanisms for synchronising the project team, for structuring their relations, and for boundary spanning. A coordination strategy contributes to coordination effectiveness, which has explicit and implicit components.

The primary contribution of this theory is an explanation of how agile software development practices act together to achieve effective project coordination. The coordination strategy concept can be used to select practices from agile methods to ensure software projects achieve effective coordination. In addition, once operationalised in future work, the well-grounded theoretical concepts developed in this research will provide valuable tools for measuring the coordination effectiveness of agile method adoption and adaptation.

ACKNOWLEDGEMENTS

My primary acknowledgement is to Sid Huff who not only guided me during the development of this thesis, but also willingly introduced me to the wider world of research. Thank you for your excellent supervision, even when you were busy as Head of School, or on the other side of the world. Thank you, Sid, for being what you are - helpful, supportive and encouraging, positive and open-minded, and always ready to share your knowledge. Not to mention diligent in reading and commenting on the many versions of this thesis, and showing me how to publish. I am grateful for your help and the value you added to my years of PhD study.

Thanks also to Sebastian Link for applying your insightful critique to make this thesis more rigorous. Beverley Hope, thank you for your good guidance, and for always being there with an encouraging word through the ups and downs of the thesis process.

Thanks to Mary Tate and David Johnstone for your open doors and ready support, and to my PhD cohorts in the School of Information Management, and the School of Engineering and Computer Science. You are too many to mention individually, but always a source of support, both academic and personal.

Thank you to Tiso Ross for smoothing my path cheerfully and efficiently through the reefs of bureaucracy. Also to the many other helpful people at Victoria University of Wellington and particularly the academic, administrative, and technical staff of the School of Information Management.

I appreciate the support given by the Agile Professionals Network in Wellington, and for their assistance in locating and introducing me to research participants. Especial thanks to Sandy Mamoli, Mike Lowery, Douglas Talbot, Tiong-Thye Goh, Allan Sylvester, and Hartmut Hoehle for using your networks to help locate participants.

Thank you to my many research participants who contributed many hours of their time to talk about their work, of which they can be justly proud.

Chris Brosnahan, thank you for your patience, good sense, and good humour, for believing I could do it, and then letting me get on with it. Thanks also to my great friends, Debbie Durant, Renata Balfour, Rachel Kidd, Ray McEnhill, and Gerard McCabe for your support and encouragement, for stepping in when things got tough, and making me take time out for a bit of fun every now and then.

TABLE OF CONTENTS

1.	INTRODUCTION	11
1.1	The Research Problem.....	15
1.2	Research Purpose and Research Questions.....	16
1.3	Research Methodology	17
1.4	Dissertation Structure.....	17
2.	LITERATURE REVIEW.....	19
2.1	The Agile Development Approach	20
2.1.1	Historical Perspective.....	20
2.1.2	Terminology in Information System Development.....	24
2.1.3	Agile Methods.....	25
2.2	Coordination	48
2.2.1	Coordination in Organisation Studies	49
2.2.2	Coordination Theory.....	53
2.2.3	Coordination in IS Project Studies.....	61
2.2.4	Coordination in Teamwork Studies	68
2.2.5	Coordination in Agile Software Development	70
2.2.6	A Summary of Coordination Perspectives.....	72
2.2.7	Project Success and Coordination.....	75
2.3	Conceptual Framework.....	89
2.4	The Research Questions in Detail.....	93
2.5	Summary	95
3.	RESEARCH METHODOLOGY	97
3.1	Research Paradigm and Approach.....	97
3.2	Overview of the Research Process.....	106
3.3	Case Design.....	109
3.3.1	Unit of Analysis.....	109
3.3.2	Case Selection.....	110
3.4	Data Collection	115
3.5	Data Analysis	124
3.5.1	Within-Case Analysis.....	124
3.5.2	Cross-Case Analysis	129
3.6	Ethical Considerations	130
3.7	Validity, Reliability, and Relevance	131
3.8	The Elements of Theory	136
3.9	Summary	140
4.	THE CASES.....	141
4.1	Overview.....	141
4.2	Case Hour: Pilot Case.....	143
4.3	Case Land	143
4.4	Case Storm	161
4.5	Case Silver.....	182
4.6	Case Rock.....	197
4.7	Summary	216
5.	A THEORY OF COORDINATION IN AN AGILE CONTEXT	219
5.1	A Coordination Strategy Concept.....	219
5.1.1	Synchronisation.....	220
5.1.2	Structure.....	224
5.1.3	Boundary Spanning	226

5.2	Coordination Strategy Antecedents.....	231
5.2.1	Project complexity.....	231
5.2.2	Project uncertainty.....	231
5.2.3	Organisation structure.....	232
5.3	A Coordination Effectiveness Concept.....	233
5.3.1	Explicit coordination effectiveness.....	234
5.3.2	Implicit coordination effectiveness.....	235
5.3.3	Coordination effectiveness in project Rock	239
5.4	Boundaries, Associations, and States.....	242
5.5	Theoretical Propositions.....	246
5.6	Summary.....	252
6.	DISCUSSION AND CONCLUSION.....	255
6.1	Answering the Research Questions.....	255
6.2	Discussion.....	269
6.3	Summary of Findings.....	273
6.4	Contributions to IS Theory	276
6.5	Contributions to IS Practice	279
6.6	Limitations of the Research	280
6.7	Future work.....	283
7.	REFERENCES	287
8.	APPENDICES.....	311
8.1	Appendix A Ethics Approval	311
8.2	Appendix B Participant Information	312
8.3	Appendix C Consent Forms	315
8.4	Appendix D Interview Protocols.....	317
8.5	Appendix E The Questionnaires.....	321
8.6	Appendix F Data Sources	333
8.7	Appendix G Glossary of Terms.....	336
8.8	Appendix H Summary of XP and Scrum.....	338
8.9	Appendix I Land Case Description	342
8.10	Appendix J Storm Case Description	342
8.11	Appendix K Silver Case Description	342
8.12	Appendix L Rock Case Description	342

LIST OF TABLES

Table 1 Agile methods by publication date	26
Table 2 Selected definitions of coordination	49
Table 3 Dependencies and coordination mechanisms (Malone & Crowston, 1994)....	55
Table 4 Grinter's (1995) dependencies in software development.....	64
Table 5 Empirical studies of project success	78
Table 6 Complexity items in ISD projects adapted from Xia and Lee (2005)	85
Table 7 Uncertainty factors adapted from Barki, Rivard and Talbot (1993)	86
Table 8 Cases and selection criteria	115
Table 9 Case description framework.....	126
Table 10 Rigor in the design phase, adapted from Dubé and Paré (2003)	133
Table 11 Rigor in the data collection phase, adapted from Dubé and Paré (2003)	134
Table 12 Rigor in the data analysis phase, adapted from Dubé and Paré (2003)	135
Table 13 Shapira's (2011) schema of theory types	137
Table 14 Overview of the four cases	142
Table 15 Land data sources	146
Table 16 Land dependencies.....	149
Table 17 Land summary of dependencies	152
Table 18 Land synchronisation activities	153
Table 19 Land synchronisation artefacts.....	155
Table 20 Land dependencies and coordination mechanisms	157
Table 21 Land dependency percentages - secondary dependencies.....	158
Table 22 Land dependency percentages - primary dependencies	158
Table 23 Land coordination mechanisms	159
Table 24 Storm data sources.....	164
Table 25 Storm dependencies	167
Table 26 Storm summary of dependencies.....	169
Table 27 Storm synchronisation activities.....	171
Table 28 Storm synchronisation artefacts.....	173
Table 29 Storm boundary-spanning activities.....	174
Table 30 Storm dependencies and coordination mechanisms	177
Table 31 Storm dependency percentages - secondary dependencies	178
Table 32 Storm dependency percentages - primary dependencies.....	178
Table 33 Storm coordination mechanism percentages	178
Table 34 Silver data sources.....	184
Table 35 Silver dependencies	185
Table 36 Silver summary of dependencies	188
Table 37 Silver synchronisation activities.....	189
Table 38 Silver synchronisation artefacts	191
Table 39 Silver dependencies and coordination mechanisms	194
Table 40 Silver dependency percentages - secondary dependencies.....	195
Table 41 Silver dependency percentages - primary dependencies	195
Table 42 Silver coordination mechanism percentages	195
Table 43 Rock data sources	200
Table 44 Rock dependencies.....	201
Table 45 Rock summary of dependencies	204
Table 46 Rock synchronisation activities	206
Table 47 Rock synchronisation artefacts.....	207
Table 48 Rock impersonal artefacts.....	208
Table 49 Rock dependencies and coordination mechanisms.....	213
Table 50 Rock dependency percentages - secondary dependencies.....	214
Table 51 Rock dependency percentages - primary dependencies	214

Table 52 Rock coordination mechanism percentages	214
Table 53 Comparison of secondary dependencies	256
Table 54 Comparison of primary dependencies	257
Table 55 Coordination mechanisms and agile practices on all projects	259
Table 56 Data sources	334
Table 57 Summary of Extreme programming.....	338
Table 58 Summary of Scrum.....	340

LIST OF FIGURES

Figure 1 Agile manifesto (Beck, et al., 2001a).....	12
Figure 2 Principles of the agile manifesto from Beck, et al. (2001a).....	13
Figure 3 Agile drivers, capabilities and value from Conboy and Fitzgerald (2007a)	43
Figure 4 Dependency relationships adapted from Crowston (2003)	57
Figure 5 Coordination and control (Sabherwal & Choudhury, 2006).....	66
Figure 6 Espinosa et al.'s (2004) framework.....	90
Figure 7 The conceptual framework.....	93
Figure 8 The research process.....	108
Figure 9 Variance and process theories from Markus and Robey (1988)	140
Figure 10 Land: stakeholders and their communication links	145
Figure 11 Land coordination strategy.....	160
Figure 12 Storm: stakeholders and their communication links	163
Figure 13 Storm coordination strategy	181
Figure 14 Silver: stakeholders and their communication links	183
Figure 15 Silver coordination strategy	197
Figure 16 Rock coordination strategy.....	217
Figure 17 Synchronisation sub-components.....	223
Figure 18 Structural sub-components	226
Figure 19 Boundary spanning sub-component.....	229
Figure 20 The coordination strategy concept.....	230
Figure 21 Coordination strategy antecedents.....	233
Figure 22 The coordination effectiveness concept	239
Figure 23 Coordination in co-located agile software development projects	246

LIST OF ABBREVIATIONS

ASD	Adaptive System Development
BDD	Behaviour-Driven Development
CAS	Complex Adaptive Systems
CVS	Concurrent Versioning System
DSDM	Dynamic System Development Method
FLOSS	Free/Libre Open Source Software
GSD	global software development
GUI	Graphical User Interface
IDE	Integrated Development Environment
IS	information systems
ISD	information system development
ISDM	information system development methodology
IT	Information Technology
JAD	Joint Application Development
LD	Lean Development
PP	Pragmatic Programming
RAD	Rapid Application Development
RUP	Rational Unified Process
SDLC	System Development Life Cycle
SE	Software Engineering
SDM	System development methodology
SMM	Shared Mental Model
TDD	Test Driven Development
TMM	Team Mental Model
UML	Unified Modeling Language
XP	Extreme Programming

1. INTRODUCTION

Software has become essential to human society. It underpins most economic activity, and increasingly contributes to social activity. Consequently, the development of software makes a significant contribution to the world economy. For example, in 2003¹, software contributed 21 billion to the UK economy, while in 2009 it contributed 180 billion to the USA economy², and made up 1% of India's 3 trillion dollar GNP³.

Although software is critically important economically and socially, the process of software development is fraught with problems. Software development is costly and complex, involves multiple stakeholders, teams of specialists, complex problems, innovative technologies, and the integration of multiple systems. Furthermore, software projects have a high failure rate. In the 1960s, the rate of failed projects was so high the situation was called a 'software crisis' (Naur & Randell, 1968), and more than 40 years later reports of software project failure persist⁴. Criticality, cost, complexity, and an unacceptably high failure rate, make better methods for developing software systems a 'holy grail' in the software development community.

Software development is central to the field of information systems, and more particularly information systems development (Bacon & Fitzgerald, 2001; Bostrom & Heinen, 1977; R. Hirschheim & Klein, 1989; Neufeld, Fang, & Huff, 2007; Nolan & Wetherbe, 1980; Sidorova, Evangelopoulos, Valacich, & Ramakrishnan, 2008). Information systems development encompasses all of the activities needed to envision, design, and create a computerised information system, and a plethora of systems development methodologies are available to guide the process (Sambamurthy & Kirsch, 2000). This variety has emerged to accommodate the various business, social, and technical environments in which development occurs (Avison & Fitzgerald, 2006b; Iivari, Hirschheim, & Klein, 2001).

One unique approach to information systems development is agile software development. Agile software development came to prominence in the late 1990s, and is now well accepted as a way to carry out software development projects (Ambler, 2009;

¹ Source: ZDNe UK: <http://news.zdnet.co.uk/software/0.1000000121.39252324.00.htm>

² Source: <http://www.marketresearch.com/product/display.asp?productid=2283003&g=1>

³ Source: <http://www.heinz.cmu.edu/research/105full.pdf> and <http://flagcounter.com/factbook/in#economy>

⁴ Source: Roger Sessions, 2009 <http://www.objectwatch.com/whitepapers/ITComplexityWhitePaper.pdf>

Dyba & Dingsoyr, 2008; West & Grant, 2010). The agile software development approach appeals to software development organisations because it promises faster delivery of working software. It also appeals to software developers because, for many, it improves the experience of software development. However, adopting an agile approach is not simple; it requires fundamental changes in philosophy, values, and software development practices and consequently has a deep impact on the people and organisations involved. These changes make agile software development a topic worthy of intense research within the information systems development field.

Agile software development is achieved by adopting an agile method; each method is made up of a unique combination of practices and techniques drawn from existing systems development methodologies and software engineering (Abbas, Gravell, & Wills, 2008; Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). The agile methods are bound into a single class of systems development methodology by their adherence to the guiding principles an 'agile manifesto', (Beck et al., 2001a). Figure 1 shows the manifesto values; Figure 2 shows the manifesto principles.

Figure 1 Agile manifesto (Beck, et al., 2001a)

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

[Return to Manifesto](#)

Figure 2 Principles of the agile manifesto from Beck, et al. (2001a)

Agile software development methods were developed to cope effectively with change during the development process. The approach is characterised by short iterative and incremental development cycles, high levels of collaboration within a development team made up of developers and customers, and a focus on mechanisms to support feedback and learning for those involved (Boehm & Turner, 2004). The customer is involved in all aspects of development, receiving working software of negotiated quality and direct business value within a short time frame. Although the various agile methods share a philosophy and certain characteristics, each individual agile method has a unique combination of practices, and each method supports a different aspect of development.

Determining the exact impact of the agile approach on software development practice is difficult because of there is wide variation in reported adoption rates ranging from 14% (Dyba & Dingsoyr, 2008), to 35% (West & Grant, 2010), to as high as 69% (Ambler, 2009) of all software development projects. What is indisputable, is that agile software development methods have grown in popularity in the last decade, and have made an important contribution to improving software development practice (Agerfalk, Fitzgerald, & Slaughter, 2009; Dingosyr, Nerur, Balijepally, & Moe, 2012). They are advocated as one of the top ten 'best practices' of IT project management (Nelson, 2007), and some suggest that adopting the agile philosophy and practices would improve project outcomes in non-IT fields (Levitt, 2011). There is a missing component in this success story; there is a dearth of useful descriptive, explanatory, or predictive theory about agile software development, and its relationship with software project success (Cao & Ramesh, 2007; Gregor, 2006).

Software project success has many antecedents. One critical factor is project coordination. One simple definition of coordination is that it is the harmonious organisation of activities carried out to achieve common goals. In information systems development, software components, teams, activities, and resources, must all be successfully coordinated. Well-coordinated development contributes to well-integrated software, produced in short time frames, with less rework, and at lower cost. (Amrit & van Hillegersberg, 2008; Andres & Zmud, 2001; Barthelmess, 2003; Chen, Jiang, Klein, & Chen, 2009; Curtis, Krasner, & Iscoe, 1988; Espinosa, Slaughter, Kraut, & Herbsleb, 2007; Grinter, 1996; Herbsleb & Mockus, 2003; Kraut & Streeter, 1995; McChesney & Gallagher, 2004; Nidumolu, 1995; Toffolon & Dakhli, 2000). Conversely, poor coordination is known to contribute to development problems (Curtis, et al., 1988).

Three areas of research are rich sources of knowledge on coordination: organisation studies (Okhuysen & Bechky, 2009), IS project studies (Kraut & Streeter, 1995; Nidumolu, 1995), and general teamwork (Mohammed, Ferzandi, & Hamilton, 2010). Each of these areas is relevant to software development practice because software development occurs in organisations within a project framework, and agile software development emphasises teamwork.

This study draws on this body of research and empirical data from software projects to explore the nature of coordination in the context of the co-located agile software development project.

1.1 THE RESEARCH PROBLEM

One of the goals of a system development methodology is to provide ways to organise and coordinate development (Avison & Fitzgerald, 2006a; Cockburn, 2002; McChesney, 1997). In the 1980s, poor coordination was a recognised problem in software development projects (Curtis, et al., 1988). In 1995, Nidumolu investigated coordination in software projects and articulated the problem: *“how can software development projects be coordinated more effectively in the presence of uncertainty?”* (Nidumolu, 1995, p. 213). Given that the agile development approach is designed to support development in changeable and uncertain conditions, a study of coordination within agile development projects is likely to contribute to answering this question.

There are different views on coordination within the agile methods community. Agile method authors consider that their methods provide flexible yet well-coordinated organisation of activities (Beck, 2000; Cockburn, 2002; Highsmith, 2000; Palmer & Felsing, 2002; Poppendiek & Poppendiek, 2003; Pries-Heje & Pries-Heje, 2011; Schwaber & Beedle, 2002; Stapleton, 1997). Some researchers support this view and consider the agile approach to be coordination intensive due to high task interdependency (Cao & Ramesh, 2007), which increases the need for coordination (Levitt et al., 1999; Van de Ven, Delbecq, & Koenig, 1976). Others consider that such development occurs most effectively in a middle ground between ‘chaos and bureaucracy’ (X. Wang & Vidgen, 2007). A contrarian view is that agile development lacks coordination, there is no ‘process’, development is chaotic and akin to ‘hacking’, and proceeds only through unstructured face-to-face communication (Stephens & Rosenberg, 2003). Although there is no consensus on coordination in agile methods,

agile methods eschew traditionally accepted ways to achieve coordination (Wagstrom & Herbsleb, 2006).

Agile methods de-emphasise traditional coordination methods such as detailed contracts, extensive planning, and documentation (Boehm & Turner, 2004).

Alternative practices are offered such as face-to-face communication, close collaboration, and a short iterative and incremental development process. Many of these practices are designed to support coordination, but existing research provides little understanding or evidence about how these practices, singly or in concert, contribute to project coordination.

Another serious issue arises because agile methods are typically adapted to suit project contingencies (Aydin, Harmsen, van Slooten, & Stegwee, 2005; Fitzgerald, Hartnett, & Conboy, 2006; Fruhling & de Vreede, 2006; Ramesh, Cao, & Baskerville, 2007). Since each project may adapt its method differently, this leads to different sets of practices within different projects. Even with extensive experience reports (Conboy & Fitzgerald, 2007b), and some rigorous studies of how agile methods are combined and adapted within projects (Fitzgerald, et al., 2006), there is no body of research exploring or explaining how agile practices provide effective coordination.

This leads to the central research problem addressed in this thesis. If the traditional means of coordination in software projects such as planning, documentation, and detailed contracts are not used in agile software projects, then what are the means of coordination, and how do these means achieve effective coordination?

1.2 RESEARCH PURPOSE AND RESEARCH QUESTIONS

The purpose of this study is to build a theory of coordination in agile software development projects. To achieve this, the research explores how software projects are coordinated when using a co-located agile development approach and how this leads to coordination effectiveness. Therefore the research questions are:

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

1.3 RESEARCH METHODOLOGY

This empirical study uses an exploratory multi-case study strategy suitable for theory building (Eisenhardt, 1989), which is a good fit with the theoretical and practical constraints of the problem domain. The selected research paradigm follows a positivist epistemology and ontology as defined by Orlikowski and Baroudi (1991), which means the research is carried out with the assumption that phenomena of interest are independently and objectively observable in a variety of contexts, and there are measurable relationships among them.

The positivist multi-case study strategy is suitable for theory building (Eisenhardt & Graebner, 2007), which is the main aim of the study. This strategy allows the research questions to be adequately addressed, accommodates the nascent maturity level of the current state of theory about agile software development, and allows for the complex nature of the problem domain. The unit of analysis is the software development project, and the primary data collection method is the semi-structured interview supported with evidence from other sources.

Information systems is an applied field so rigor and relevance are equally important considerations. Paré's (2004) guidelines for rigor in positivist case study research are followed. Relevance is achieved by obtaining data from on-going projects, and involving participants in data verification (Roseman & Vessey, 2008).

The overall study design follows the steps for theory building from exploratory multi-case study research defined by Eisenhardt (1989) and Paré (2004). Cases are obtained from organisations located in Wellington city, New Zealand and include a single pilot case, plus four cases from different organisations. Data analysis follows the general heuristics of Crowston and Osborn (2003) for identifying dependencies and coordination mechanisms in work practices, supported by qualitative data analysis techniques developed by Miles and Huberman (1994) and Thomas (2006).

1.4 DISSERTATION STRUCTURE

This chapter has introduced agile software development and coordination, defined the research problem, summarised the research goal, and presented research questions and the research methodology. Further chapters contributing to this thesis are as follows.

Chapter 2 Literature Review

The literature review chapter reviews research into agile methods and coordination, and identifies coordination concepts relevant to agile software development. The final sections present the guiding conceptual framework and the research questions in detail.

Chapter 3 Research Methodology

This chapter defines the research methodology by explaining the research paradigm and justifying the selection of a case study approach. Sections include the research design, case selection, data collection, and data analysis procedures. Validity, reliability, relevance, and ethical considerations are included. The final section discusses the elements of theory that are used to structure the presentation of the theory developed in Chapter 5.

Chapter 4 The Cases

This chapter discusses the pilot case and presents within-case analysis of each of the four cases. Each within-case analysis consists of:

- A detailed case description organised using a pre-defined framework.
- An analysis of the coordination strategy of the case project including the influence of project complexity, project uncertainty, and other antecedents.

Chapter 5 A Theory of Coordination in Agile Projects

This chapter develops and presents the theory of coordination in co-located agile software development projects. This chapter defines the concepts of coordination strategy and coordination effectiveness, and the relationship between them. The relationship is presented in the form of associations, boundary conditions, states, and propositions.

Chapter 6 Discussion and Conclusion

This chapter answers the research questions, discusses the relationship of the findings with extant literature, summarises the findings, and explains their contribution to theory and practice. Limitations of the study are discussed. The thesis concludes with a discussion of future research opportunities.

2. LITERATURE REVIEW

This literature review addresses the two major topics of this study: agile software development, and coordination. The review begins with a brief history of system development methodology, because agile software development is a class of system development methodology. The relatively small body of research into agile software development is presented next. The focus of the review then moves to coordination.

Four streams of research on coordination are pertinent to agile software development: coordination in organisation studies, coordination as defined in an interdisciplinary Coordination Theory, coordination in information systems projects, and coordination in teamwork studies. Research in each of these areas is reviewed, followed by research that applies coordination theories in the context of agile software development. Because the effectiveness of coordination is another key aspect of this research, literature linking coordination and project success is also reviewed. This chapter concludes by presenting the initial conceptual framework, and the research questions that guide and bound this study.

This literature review has the following structure:

- The agile development approach
 - Historical perspective
 - Terminology in information system development
 - Agile methods
- Coordination
 - Coordination in organisation studies
 - Coordination Theory
 - Coordination in IS project studies
 - Coordination in teamwork studies
 - Coordination in agile software development
 - A summary of coordination perspectives
 - Project success and coordination
- Conceptual framework
- The research questions in detail
- Summary

2.1 THE AGILE DEVELOPMENT APPROACH

To understand agile software development and how it arose, it is necessary to understand something of the history of system development methodologies. The following section presents this history in brief. The next section provides a discussion to clarify the confusing terminology in this area. Then, agile software development, its history, and research streams pertinent to this study are addressed.

2.1.1 HISTORICAL PERSPECTIVE

Information systems development is fundamental to the discipline of information systems (Bacon & Fitzgerald, 2001; Baskerville & Myers, 2002; Rudy Hirschheim, Klein, & Lyytinen, 1996; Neufeld, et al., 2007). For example, Sidorova states *“The Information Systems academic discipline focuses on how IT systems are developed and how individuals, groups, organisations, and markets interact with IT”* (Sidorova, et al., 2008, p. 475).

Information systems as a research field draws upon reference disciplines (Baskerville & Myers, 2002); this study draws on various disciplines including software engineering where the agile methods are a focus of intense interest (Dyba & Dingsoyr, 2008). Researchers from both information systems and software engineering publish together on this topic because agile software development affects both the social and technical milieu. Research in this area covers system architecture, system quality, programming practices, and development tools, as well as, teamwork, project management, project success, contract development, human resource management, end-user involvement, and end-user satisfaction.

The key difference between studies of system development in software engineering and information systems is that IS researchers study agile methods from the perspective of existing IS or organisation theory, or build new theory to explain their findings (Morrison & George, 1995). This is much less common in software engineering studies where behavioural research is not the central focus (Dingsoyr, Dyba, & Abrahamsson, 2008; Glass, Ramesh, & Vessey, 2004; Seaman, 1999). As most of the early research into agile software development emerged from the software engineering field, this section of the literature review on agile software development draws extensively on software engineering research.

Systems development methodologies (SDM) began to appear in the late 1960s after a ‘software crisis’ was identified (Naur & Randell, 1968). The software engineering field

coined this term when large-scale development was proving problematic. The issue was described this way, “*certain classes of systems are placing demands on us which are beyond our capabilities and our theories and methods of design and production at this time*” (Naur & Randell, 1968, p. 10). By the 2000s, a plethora of systems development methodologies was available to guide system development. Iivari et. al (2001) and Avison and Fitzgerald (2006b) provide extensive overviews of these methodologies.

No single widely accepted definition of ‘systems development methodology’ is available and definitions have changed over time. One commonly accepted definition is:

A systems development methodology is a recommended means to achieve the development, or part of the development, of information systems based on a set of rationales and an underlying philosophy that support, justifies and makes coherent such a recommendation for a particular context. The recommended means usually includes the identification of phases, procedures, tasks, rules, techniques, guidelines, documentation and tools. They might also include recommendations concerning the management and organisation of the approach and the identification and training of the participants. (Avison & Fitzgerald, 2006a, p. 568)

During the 1980s, a number of conferences in information systems were devoted to defining, analysing, and comparing methodologies, primarily system design methodologies. At that stage methodologies were seen as a way to bring control and repeatability to the development of automated business systems (Olle, Sol, & Tully, 1983). These conferences advocated the use of a methodology to reduce the inherent risk in system development, and to document mechanisms for carrying out systems analysis and design activities. At this time, system development was viewed as consisting of sequential phases typically comprising, analysis, design, development, and testing. Each phase of development was treated separately requiring different techniques, skills, and modelling methods. Methodologies were created to support what was called the ‘structured’ approach and were designed to support the structured programming languages and relational databases available (e.g. Information Engineering (Martin, 1989), Yourdon Systems Method (Yourdon, 1989), SSADM (Eva, 1994)). Many methodologies followed the process logic elaborated by Royce in 1970 in what became known as the System Development Lifecycle (SDLC) or waterfall model (Royce, 1987). By 1994 thousands of systems development methodologies were available, some in the public domain and others developed in-house (Jayaratna, 1994). Further technical developments brought object-orientation; and a new group of methodologies were created based on object-oriented modelling techniques and development languages (e.g. Rational Unified Process (Kruchten, 2000), OPEN

(Graham, Henderson-Sellers, & Younessi, 1997)). Yet other approaches focused on the end-user aspects of systems development such as the Participatory Design movement (Kuhn & Muller, 1993). Problem solving aspects of systems design were addressed by methodologies such as Soft Systems Methodology (Checkland, 1999).

By the 1980s and 1990s, methodologies for all aspects of development were available, however a parallel stream of critique and empirical research reported the failure or misuse of methodologies in practice. Parnas and Clements (1986) advised the software engineering community to ‘fake it’. Rather than follow the ideal, that is the rational, systematic way to develop a software system proposed by SDLC-based methodologies, which involves writing requirements and design documents before beginning development, developers should construct the necessary documentation as system details emerge during development. In this way, the final versions of documents describing requirements, design decisions, and software modules would match the final version of the delivered system. Avgerou and Cornford (1993) criticised system development methodologies because they thought they had serious negative consequences for practitioners as they did not allow for experience, judgment, and creative thinking. They advocated permissive methodologies to allow “*space for practitioners to make their own decisions*” (Avgerou & Cornford, 1993, p. 285). Further research found methodologies were sometimes used as a social defence, that is, “*Methodology, whilst masquerading as the epitome of rationality, may thus operate as an irrational ritual, the enactment of which provides designers with a feeling of security and efficiency at the expense of real engagement with the task at hand.*” (Wastell, 1996, p. 25). Nandhakumar and Avison (1999) found the proposed benefits of following a traditional IS development methodology (Information Engineering, in that study) were a fiction. After a field study of a large-scale system development, they concluded that “*traditional methodologies are too mechanistic to be of much use in the detailed, day-to-day organization of developers’ activities*” (Nandhakumar & Avison, 1999, p. 187). Further studies found methodologies were not used in their published form (Fitzgerald, 1997, 2000; Goulielmos, 2004; Karlheinz Kautz, Hansen, & Jacobsen, 2004; Westrup, 1993) and in many cases, not used at all (Truex, Baskerville, & Travis, 2000).

One outcome of this rejection of system development methodologies as solutions to the problems of development was the idea of method engineering (Truex, et al., 2000). In method engineering, development techniques from different methodologies are formally pre-selected to create a unique methodology tailored for the situation in which the development occurs (Brinkkemper, 1996; Henderson-Sellers, 2003; Kumar &

Welke, 1992; van Slooten, 1996). This approach does suffer from problems, such as the need for extensive knowledge of the methodology base before selections can be made, and the need for knowledge repositories (Fitzgerald, et al., 2006).

Given these issues with methodology use, in the 1990s, research turned to the study of systems development using qualitative research methods to investigate how development occurs in practice (Fitzgerald, 2000; Lyytinen & Rose, 2003b; Russo & Stolterman, 2000; Wynekoop & Russo, 1995, 1997). Driving this research was the knowledge that, in practice, methodologies were seldom used as specified, so taking a system development methodology lens to the study of practice was not productive. Key findings from Fitzgerald (2000) based on his series of quantitative and qualitative studies of systems development during the second half of the 1990s, indicated that due to the increasing pace of the business environment, and a shift toward adopting packaged software and outsourcing, new 'canons' for development were needed. These included the need for adoption of practices such as time boxing (i.e. fixed length iterations), 'frequent tangible returns' (i.e. incremental development), and a mix of traditional linear process approaches and rapid application development approaches based on the situation rather than an explicit methodology. Fitzgerald (2000) also advocated for the development of 'good enough' systems produced within an appropriate time scale, 'flexible' development processes, and empowerment of developers by allowing them more discretion as to how they carry out development. Further, he proposed developing methodologies as frameworks that define only high-level goals and deliverables.

Another key series of qualitative studies by Lyytinen and Rose (2003a, 2003b) supported Fitzgerald's (2000) conclusions. Their results, based on insights from eight system development organisations creating Internet based systems, showed that developers wanted less rigid methodologies to allow for changes in the way that projects were planned, controlled, and coordinated when accommodating a drastically reduced delivery time for software. In these studies, developers stated that methodologies available at that time were inappropriate for Internet development.

This brief history of system development methodologies up until the late 1990s shows a change in thinking within the IS development research community, from assuming methodology use is universally beneficial, to a recognition that methodology use can be problematic. It also shows that the business environment, with its faster pace, and new technology environments, such as object-orientation and the internet, influence

methodology creation. Further, methodologies are an ‘ideal’ that is seldom achieved in practice, and there is no one universal methodology appropriate for all types of development. In the late 1990s, agile methods emerged to contribute to this landscape. Before proceeding with the literature review, the next section attempts to clarify some of the confused terminology in information systems development relevant to a detailed presentation of agile methods and their literature.

2.1.2 TERMINOLOGY IN INFORMATION SYSTEM DEVELOPMENT

There is considerable confusion surrounding the terminology in information systems development studies with respect to the terms methodology, method, technique, and practice. This is because usage differs on different continents, and within different research streams. This section discusses how this study uses these terms.

Methodology in the context of system development is the first problematic term. For example, Iivari, Hirschheim and Klein (2001) discuss how they specifically avoid the word ‘methodology’ when naming the levels in their comprehensive analytical framework of system development methodologies (discussed in the next section). A number of authors in the field of information systems development have discussed this issue (Cronholm & Agerfalk, 1999; Wynekoop & Russo, 1995) and, even though Jayaratna states “*the term methodology is pragmatically well established within the field of information systems to mean the same as method*” (Jayaratna, 1994, p. 14), there is no consensus.

There are two perspectives on the meaning of methodology, method, and technique (Wynekoop & Russo, 1995). One view is that methodology is the study of methods. A method (e.g. the agile ‘method’ Extreme Programming, or XP) contains a number of interrelated techniques. A technique is a detailed procedure for carrying out some activity (e.g. create a class diagram, perform pair programming, or develop a story card). The alternative view uses the word ‘methodology’ to mean ‘method’, so for example, XP is a methodology and the methodology consists of techniques such as pair programming.

There is also a problem with the word ‘method’; in many instances, this word means the same thing as ‘technique’ and the two terms are used interchangeably.

To add to the terminology confusion the term ‘process’ is used in software engineering to mean two different things. Boehm defines a process model as describing the

ordering of stages involved in software development. A process model addresses the questions: “(1) *What shall we do next* (2) *How long shall we continue to do it*” (Boehm, 1988, p. 61). In general then, a process implies time ordering of events. Other researchers give ‘process’ the same meaning as ‘methodology’. Thus a process should include a philosophy, guidelines, phases, techniques, tools (Graham, et al., 1997) and all the other parts of a methodology as defined in section 2.1.1 Historical Perspective. Thus, XP would be defined as a ‘process’.

The term ‘practice’ is an umbrella term commonly used to refer to any technique or commonly accepted way of working within the agile development community. Examples from XP include things such as ‘maintaining a 40 hour work week’, ‘customer-on-site’, ‘refactoring’, or ‘pair programming’.

In this study the following definitions are used. An agile method (e.g. Scrum or XP) is a unique systems development methodology, following the definition of methodology provided in section 2.1.1 Historical Perspective. Agile methods form a distinct class of methodologies or unique approach to software development following Iivari et al.’s (2004) classification. Each agile method has its own unique process, following Boehm’s (1988) definition, and in general that process will be iterative and incremental rather than sequential, as in the SDLC-based methodologies. Furthermore, each agile method is made up of a unique set of practices, as defined in the paragraph above, although some practices are common to more than one agile method. The final term used is ‘agile software development’, and this term is applied whenever any agile method or mix of agile method practices is in use.

2.1.3 AGILE METHODS

Individual agile methods were created in reaction to persistent problems in software development not adequately addressed by traditional system development methodologies or software engineering techniques, and the need to expedite software development in the business and technology environment of the late 1990s and early 2000s (Beck, 2000; Cockburn, 2002). Agile methods share a common basis in the practical experiences of software engineers, and ideas from new product development literature such as Takeuchi and Nonaka’s (1986) work identifying how to best manage projects when developing new products under intense time-pressure.

Initially agile methods were called ‘lightweight’ to distinguish them from ‘heavyweight’ methods. Heavyweight methods got this name because they produced many non-

software artefacts during development in the form of documented plans, models, and specifications. Examples of heavyweight methodologies are SSADM (Eva, 1994), Information Engineering (Martin & Finkelstein, 1981), Unified Software Development Process (Jacobson, Booch, & Rumbaugh, 1999), and OPEN (Graham, et al., 1997).

A definitive list of all agile methods is not available. Table 1 shows those Dyba and Dingosyr (2008) considered to be agile methods after an extensive review of the literature. After that review, Agile UP was published so it is also included in the list. The most commonly adopted agile methods are XP and Scrum (West & Grant, 2010), therefore summaries of those methods are provided in Appendix H Summary of XP and Scrum.

Table 1 Agile methods by publication date

	Agile Method	Acronym	Key source
1	Dynamic Systems Development method	DSDM	DSDM (<i>Dynamic Systems Development Method, Version 2</i> , 1995) Stapleton (1997)
2	Crystal methods	Crystal	Cockburn (1998) Cockburn (2002)
3	Extreme Programming	XP	Beck (1999) Beck (2000) 1 st Edition Beck and Andres (2005) 2 nd Edition
4	Adaptive Software Development	ASD	Highsmith (2000)
5	Scrum	Scrum	Beedle, Devos, Sharon, Schwaber, & Sutherland (1999) Schwaber & Beedle (2002)
6	Feature Driven Development	FDD	Palmer & Felsing (2002)
7	Lean Development	LD	Charette (2002) Poppendiek & Poppendiek (2003)
8	EVO	EVO	Gilb (2005)
9	AgileUP	AUP	Ambler (2008) First published on-line 2005

A manifesto for agile software development was published in 2001 setting down the values and principles which the manifesto authors believed were common to all agile

methods (Beck, et al., 2001a). The manifesto authors were seventeen people who had conceived and published individual lightweight methods with similar characteristics. Each method was based on practitioner experience, evolutionary development practices, and the idea of early delivery of quality software. During development of the manifesto, lightweight methods were renamed agile methods.

Agile methods were designed for development to occur in small, co-located teams of up to 10 developers. Software project involved creating small internet-based applications and interface-intensive systems in situations where requirements uncertainty and speed-to-market were overwhelmingly important (Beck, 2000; Strode, 2005; Schwaber & Beedle, 2002).

Many pre-existing ideas, practices, and techniques from earlier software engineering and information systems movements and methodologies contributed to individual agile methods (Abbas, et al., 2008; Abrahamsson, et al., 2003; Hilkka, Tuure, & Matti, 2005). Abrahamsson et al.'s (2003) investigation of the evolution of agile methods found the main influences to be object-orientation, evolutionary development, Internet technologies, and method engineering. The object-oriented influences came from the Unified Modeling Language (UML) (Rumbaugh, Jacobson, & Booch, 1999) and the Rational Unified Process (RUP) (Kruchten, 2000). Evolutionary approaches include the influence of Boehm's spiral model (Boehm, 1988), prototyping, and Rapid Application Development (RAD) (Martin, 1991). The Internet technologies include ideas from Microsoft's Sync and Stabilise method (Cusumano & Selby, 1997), and Internet speed development (Cusumano & Yoffie, 1999). Method engineering ideas came from Kumar and Welke (1992) and 'amethodical' development (Baskerville, Travis, & Truex, 1992; Truex, Baskerville, & Klein, 1999).

Influences from information systems include the Participatory Design movement and ideas from soft systems methodology. Participatory Design (Kuhn & Muller, 1993) is a Scandinavian movement focussing on stakeholder participation in the development of information technology solutions. The influence of Checkland's soft systems methodology (Checkland, 1999) and ideas of human activity systems being holistic and evincing emergent properties appear in Crystal methods (Cockburn, 2002) and ASD (Highsmith, 2000).

Agile methods generally conform to a profile developed by Strode (2006) after analysing five agile methods including XP, Scrum, DSDM, Crystal methods, and ASD:

An agile method is a software development methodology designed for the management and support of iterative and incremental development of business systems in environments where change is constant. Agile methods use software development techniques that enhance teamwork in small, empowered teams and support active customer involvement. An agile method is designed to produce working software early, using communication, feedback, learning and frequent meetings in preference to modelling and documentation. Agile methods adapt existing software development techniques to achieve these goals. (Strode, 2006, p. 262)

Each agile method is a unique system (or software) development methodology, according to the definition of Avison and Fitzgerald (2006a, p. 568) (provided in section 2.1.1 Historical Perspective), and each agile method has a different purpose. For example, XP is specifically designed for software development in high change environments, for satisfying customer needs, and for maintaining effective teams (Beck, 2000). Scrum focuses on project management of iterative development (Schwaber & Beedle, 2002). DSDM is a framework for Rapid Application Development (RAD) (Stapleton, 1997). Crystal methods provide techniques for designing a methodology to suit a specific project (Cockburn, 2002), and Adaptive System Development (ASD) is a framework for managing software projects under intense time pressure (Highsmith, 2000).

Iivari, Hirschheim, and Klein (2004) classify agile software development methods as a unique *approach* to information system development (ISD). That is, agile methods together form a unique class of methodologies. These authors organised information systems development process knowledge into a hierarchical framework of categories including paradigms, approaches, methods, and techniques. *Paradigms* are at the highest level; they are related to philosophical assumptions and take into account ontology, epistemology, methodology, and ethics. At the next level are *approaches*, which take into account the goals, guiding principles, fundamental concepts, and principles of an ISD process. Approaches include items such as systems analysis and systems design, information management, socio-technical design, infological design, object-orientation, interaction design, soft systems methodology, agile methods, and open source software. The *method* level defines the relationship between techniques and provides detail on the ISD processes. Examples of methods, in this framework, are Information Engineering, ETHICS, object-oriented software engineering, DSDM and XP. The *techniques* level includes the detailed concepts and notations of each method. Items at this level include activities and models such as rich pictures, use case models, pair programming, and class diagrams.

This introduction to agile methods has shown that although they are strongly associated with earlier methodology and process movements and draw heavily on existing practices in software engineering, each agile method is unique and together they form a unique approach to system development. What distinguishes them from other methodologies is their overall philosophy espoused in the values and principles described in the agile manifesto.

The next section reviews the systematic reviews of agile software development literature to provide an overview of the state of research on this topic.

Reviews of Agile Software Development Literature

Two major reviews of agile software development research were published in 2002 and 2008. The first, by Abrahamsson, Salo, Ronkainen and Warsta (2002), integrated the knowledge and terminology on this topic. Their review was based mainly on information in the books published on individual methods, because there was so little research available at the time; agile methods had only been formally named in 2001 (Beck, et al., 2001a). Consequently, Abrahamsson, et al. (2002) found no rigorous empirical research; anecdotal reports from adopters were all that was available. They compared 10 methods they considered could belong to the agile approach. They concluded that agile methods take a people-centric view of software development, they are designed for use in small teams, they uniquely amalgamate existing practices from software engineering, and they are unlikely to be suitable for all types of project. They also noted that scaling the practices of agile methods to work effectively in large projects was viewed as a major problem in their adoption because they were designed for projects with teams of 10 or fewer developers.

Building on this review, Abrahamsson, Warsta, Siponen and Ronkainen (2003) went on to compare nine agile methods using an analytical framework consisting of lifecycle coverage, project management support, whether the method offered abstract principles or concrete guidance, and the universal-applicability or situation-appropriate nature of the method. They concluded that agile methods do not offer complete lifecycle coverage, they lack techniques for project management and, apart from XP and Pragmatic Programming (Hunt & Thomas, 2000), they offer only philosophy and values, rather than explicit techniques. They also found that Feature Driven Development and Crystal Methods are the only agile methods that claim to be universally applicable. The others claim to be situation specific or tailorable to suit

different projects, although the published methods provide little advice on how to carry out this customisation.

The second major review, based on a systematic search of all major academic databases and conference proceedings in the software engineering field in the period from 2001 to early 2005, was carried out by Dyba and Dingosyr (2008). Over that period, they found only 33 rigorous empirical studies, of which 75% investigated aspects of XP (XP was the most commonly used agile method in that period). They found no empirical studies before 2001, and the majority of research was published in conference proceedings, because this is the preferred avenue for publishing by software engineering researchers. Information systems journals are almost silent on the topic during the period covered by their review.

This review by Dyba and Dingosyr (2008) contributed in three ways to a better understanding of agile software development. First, they identified the major topics in agile software development research including adoption, development process, project management, knowledge management, organisational culture, collaborative work, team work, customer studies, developer studies, student studies, productivity, product quality, work practices, and job satisfaction (Dyba & Dingsoyr, 2008).

Dyba and Dingosyr (2008) also argued that the research base was nascent, based on the definitions provided by Edmondson and McManus (2007). These authors categorise field research as nascent, mature, or intermediate depending on the maturity of the theory base on which the research builds. Nascent research has little or no prior theory, mature research has existing precise models that can be extended or linked with other theory, and intermediate research is in a state where some theoretical constructs may be well defined and others less so. Dyba and Dingosyr (2008) noted that, although holistic studies of agile methods are at the nascent stage, research into individual practices used within agile methods (e.g. pair programming, refactoring) are at an intermediate stage. They concluded that theory building, rather than theory testing, is more appropriate while agile software development theory maturity is in the nascent stage (this is discussed in depth in Chapter 3 Research Methodology).

The third contribution of Dyba and Dingosyr's (2008) review was to provide a summary of areas for further research. They found a clear need for rigorous empirical studies and called for studies of mature teams, rather than adoption studies, to identify successfully embedded practice. They also identified a need for more empirical studies into the effectiveness of Scrum as there were few studies, its use was growing within

industry (confirmed in a survey by West and Grant (2010)), and this method has important implications for project management practice.

Since that review, the information systems field has begun to provide theory-building and theory-testing studies of agile methods. Examples include, a theory testing study of the impact of the agile approach on system use (Hong, Thong, Chasalow, & Dhillon, 2011), a theory developing understanding of the interplay between the agile approach and organisation culture (Iivari & Iivari, 2011), and guidance on adapting the agile approach for use in large mature organisations (Barlow, Giboney, Keith, Wilson, & Schuetzler, 2011). Further research on globally distributed agile software development is also now available (e.g. Agerfalk, et al., 2009). Some of this work is reviewed later in this literature review in the section on coordination in agile software development.

In summary, these major reviews of agile software development indicate that research into this approach to information system development is scant, although emerging, theory development is nascent, and many open questions remain.

The following sections of this review of agile software development literature are organised to cover ethnographic studies, studies of iterative and incremental development, studies reporting spontaneously occurring agile practices, and scalability and tailoring studies. Following this are two sections on theory related to agile development. One addresses the definition of 'agility', and the other addresses research linking existing theories with agile development. Because this study explores coordination in the agile approach, the following sections review agile software development literature while highlighting findings related to coordination.

Ethnographic Studies

Ethnography is a qualitative research methodology founded in anthropology. In ethnographies, research questions are seldom pre-specified and researchers immerse themselves in the research context, writing rich descriptions of the social situations they find. Ethnographic research has proved useful for studying information system development because it provides detailed insights into social and organisational activities. A handful of ethnographic studies have been conducted within agile development projects.

An early ethnographic study of DSDM, a form of Rapid Application Development (RAD), is that of Beynon-Davies, MacKay and Tudhope (2000) who reported on data gathered in 1996. RAD was developed in 1991, and was designed to cope with changing

requirements during development (Avison & Fitzgerald, 2006a). The method uses an iterative and incremental lifecycle with prototyping and other techniques for coping with requirements change. DSDM was initially designed as a RAD framework (Stapleton, 1997), and many practices from DSDM and RAD have migrated into the agile approach.

Beynon-Davies, et al. (2000) were the first to use ethnographic methods to study RAD in the context of intranet development. First they developed a theoretical model of RAD development practice based on normative guidelines (Beynon-Davies, Carne, Mackay, & Tudhope, 1999), and then compared observed practice with those guidelines. They studied a developer-user team isolated from their normal workplaces and required to use RAD to develop an intranet for their company within three weeks. The authors commented on coordination in the project:

To enable co-ordination of work, there was much use of time-management and co-ordination techniques. Whereas in the phased type of RAD project, the co-ordination of activities has to rely on more formal types of communication, such as documented designed minutes and memos, on this project 'the collective memory' was negotiated in relation to informal and tacit understandings relating to work co-ordination. In this sense the to-do lists and wash-up sessions served as artifacts that enabled the co-ordination of work. (Beynon-Davies, et al., 2000, p. 213)

Further ethnographic studies reported similar findings. A series of ethnographic studies of mature XP teams identified a number of coordinating activities including frequent meetings, pair programming, refactoring, and automated builds (Robinson & Sharp, 2004; Sharp & Robinson, 2004; Sharp & Robinson, 2008). These studies also found artefacts contributed to coordination, in particular the information workspace; which is an easily observed wallboard for displaying project information such as story cards (coloured cardboard cards that contain user requirements or technical requirements). Sharp and Robinson (2008) studied six teams over five years and concluded that the wall and the cards are powerful mechanisms for supporting progress tracking, collaboration and co-operation within a team. The cards, with their notation, colours, and position on the wall provided information about which requirements achieve which system functions, the dependencies between requirements, and which requirements belong to which iteration. Sharp, Robinson and Petrie (2008) concluded that "*the Wall shapes, mediates and manages the life of developers*" (Sharp & Robinson, 2008, p. 8).

Earlier research supports the finding of Sharp, Robinson and Petrie (2008). Predating the formal publication of most agile methods, an ethnographic study by Whittaker &

Schwarz (1999) of paper and electronic tools for communicating project information in co-located software teams, showed that paper based tools provide better support for synchronous interpersonal communication in complex and long term collaborative work. They found that use of a large wallboard in a public area to display planning and progress information encouraged responsibility, commitment, and more reflective planning with timely updating compared to electronic support for equivalent information display.

An ethnographic study by McKenzie and Monk (2004) found the use of a concurrent versioning system (CVS) (also called a code repository) enabled coordination within an XP project. These systems ensure that developers cannot work on code concurrently, and thus introduce errors and unnecessary rework, because code must be 'checked-in' and 'checked-out' of the system in a controlled manner. They also linked coordinative practices with the ability to accommodate change, although they did not explain how this occurred:

The artifices used to co-ordinate the work of software development in XP – unit tests, card games, code repository – directly address the problems of coordinating work against an unstable background of economic and organisational change. (MacKenzie & Monk, 2004, p. 115)

Moe, Dingsoyr, and Dyba (2008) performed a participant-observation study of an unsuccessful Scrum adoption focusing on autonomy in self-organising teams. One minor finding was that the daily stand-up meeting and autonomy over project decisions enabled coordination on that project.

While coordination is not a specific focus in these studies, they suggest that certain activities and artefacts used within agile software development seem to have the purpose of supporting effective coordination.

Iterative and Incremental Development

Agile software development is distinguished by its iterative and incremental nature (Abbas, et al., 2008; Turk, France, & Rumpe, 2005). Iteration is the repetition of one or more phases, or a set of activities, within the total development timeframe. Incremental development is the addition of small amounts of functionality to the overall system. Iteration and increments work together because every iteration progressively elaborates the system under construction by the addition of increments of functionality.

Iterative and incremental development has a long history predating agile methods. Larman and Basili (2003) traced iterative development to the 1930s, but it was when Royce (1987) presented his model of the software development lifecycle in 1970 that the idea became widely acknowledged. Once Royce's ideas were published two approaches to system development emerged: the traditional system development lifecycle approach (also named the waterfall approach), and the iterative lifecycle approach. The traditional approach involves a sequential single pass development process and it has influenced numerous system development methodologies (e.g. Yourdon Systems Method (Yourdon, 1989), Structured System Development Method (Eva, 1994)). The iterative approach involves repetitive cycles of development and delivery of functionality. Over the last 30 years, iteration duration has shortened considerably. For example Boehm's spiral model embodies an iteration period of up to 6 months (Boehm, 1988), the object-oriented methodology OPEN utilises a 3 month iteration (Graham, et al., 1997), XP has a one to four week iteration (Beck, 2000), and Scrum has a 30 day iteration called a sprint (Schwaber & Beedle, 2002).

The management of this type of development was discussed at the OOPSLA (annual conferences on Object-Oriented Programming, Systems, Languages and Applications) workshops in the early 1990s (Coplien, Hutz, & Marykuca, 1992; Mead, 2001). Participants reported that object-oriented development is effectively supported by iterative and incremental development, but there is a conflict with the traditional project management process, which is predicated on a sequential linear set of project lifecycle phases (Kerzner, 2003). Object-oriented development works well with iterative and incremental development because of the component-based nature of object-oriented development. Because these systems can be divided easily and robustly into small components, this makes both the design of an increment and the allocation of increments to iterations simpler than in non-object based systems. Agile methods were originally designed for creating systems based on object-oriented technologies (Beck, 2000; Cockburn, 2002; Highsmith, 2002; Schwaber & Beedle, 2002; Stapleton, 1997).

Evidence for the effectiveness of iterative development in uncertain and dynamic business environments was found by McCormack, Verganti and Iansiti (2001) when they examined the characteristics of an effective flexible development process. They first carried out qualitative interviews with project managers to characterise the development process, and identify and operationalise constructs related to process flexibility. They found that in software development for the Internet a more flexible, iterative and incremental process was used, as opposed to traditional sequential

phases. They also found four concepts important for effective flexible development: investment in early architectural design, early feedback from the customer, early technical feedback (i.e. early system integration), and team member's generational experience (i.e. how many completed projects the members had previously worked on). In a second phase of the study, they surveyed participants in 29 completed projects in 17 organisations to investigate the relationship between the four concepts and software product performance. Project resources and scale (size) were control variables. The value of the dependent variable, product performance, was determined using a two-round Delphi study comprising 14 industry observers. The final product performance value was calculated by taking the mean scores of the assessments of these observers.

Results showed a positive relationship between the use of flexible development processes and enhanced product performance. Early market feedback was the most important indicator for product performance. Investment in architectural design was also statistically significant. Less important was early technical feedback, and no statistical significance could be attributed to generational experience (although the author's do not discount this construct, and propose that generational experience may impact projects by reducing resources rather than improving quality). The four constructs and resource control accounted for over 50% of the variance in product quality.

The authors discussed early market feedback and the architectural challenges this creates. Early market feedback is achieved by providing an early, even if incomplete, software product to the customer, focusing on the customer's feedback, and then collaborating with the customer to co-evolve the product. The challenge in this form of development is to create a system architecture that is flexible early in the project and remains flexible as the project progresses. Based on these arguments, the authors concluded that investments in architecture are a key to enabling a flexible process.

The qualitative phase of their study provided one finding, which led them to suggest a change in mindset is needed when working with a flexible process. They illustrated this point with an example. A senior project manager reported on two projects. The manager perceived that one project was successful and the other less successful because it was 'poorly executed'. Examination of the data showed that the apparently poorly executed project accommodated continual change and that the final product had a high quality rating. To the project manager this project appeared to be chaotic and

therefore less successful. The more successful project, as perceived by the project manager, followed a controlled process, closely met its initial specifications, and according to traditional development practices, this process was much superior. What the data showed was that, because the design was frozen early in the project the final product was not well received by the time it reached the market.

In summary, this literature shows that the agile approach uses an iterative and incremental process that differs from previous iterative and incremental methods due to the very short iteration cycle. Research indicates that this process is effective and enhances flexibility; however, this conclusion is based on a single research study of large-scale Internet software development projects.

Spontaneously Occurring Agile Approaches

Research into software development projects shows that certain agile development practices arise naturally when projects are under time pressure and change is frequent.

Focusing on Internet computing as a disruptive IT innovation during the dot com era Lyytinen and Rose (2003a, 2003b) studied the relationship between system development innovation, IT base innovation, and service innovations in a confirmatory multi-site case study in eight system development organisations in Finland. Seven firms with 'rigid' methodologies reported *"they now either had to give up or radically alter their routines of how to plan, coordinate, and control development through methodologies"* (Lyytinen & Rose, 2003b, p. 572). A move to lightweight processes with prototyping was common. This was directly attributed to development for the Internet and the shortened development time this imposed: *"in e-business it's six to nine weeks"* (Lyytinen & Rose, 2003b, p. 572). Another identified trend was the move to component-based development, and the increased complexity of development. Participating firms acknowledged difficulties with coordinating the new skill sets and specialised tasks required for this form of work. In concluding their study, Lyytinen and Rose (2003b) noted a need for development methodologies that are appropriate for Internet computing, which support *"simplicity, agility, and concern for flexibility"* (Lyytinen & Rose, 2003b, p. 580).

This form of development was named short cycle time development after a series of studies of commercial Internet software development in small niche companies identified a recurring group of five characteristic practices (Baskerville, Levine, Pries-Heje, Ramash, & Slaughter, 2001; Baskerville & Pries-Heje, 2004; Ramesh, Pries-Heje, & Baskerville, 2002). These practices include a focus on completion speed, release-

oriented parallel development with prototyping, the criticality of architecture, negotiable product quality, and an ideal workforce. Initially noting that much of the work in the area was normative in focus and possibly inadequate, Baskerville and Pries-Heje (2004) took a grounded theory approach to explore software development practice. Open-ended and semi-structured interviews of 45 personnel from all organisational levels informed the study. The unit of analysis was a region, and three Danish companies and nine USA companies participated. The authors determined that short cycle time development is a distinctive new form of development (although Kautz, Madsen & Norbjerg (2007) dispute this, citing historical precedents). They proposed that this new form of development, which arose in the dot com era but has outlived that time, is driven by the need for innovation, and the efficient delivery of that innovation to the marketplace. They linked short cycle time development to the need for agile organisations that can sense and respond rapidly to market forces, and the rise of methods for fast software product development, namely XP and Scrum.

Not only is short cycle time development a distinct new form of development, but Baskerville and Pries-Heje (2004) also considered it suggests a new metaphor for systems development, that of 'gardening' or growing systems, rather than the traditional metaphor of 'engineering' or building systems.

In identifying short cycle time development, Baskerville and Pries-Heje (2004) noted two issues related to coordination. Firstly, they stressed the importance of architecture as a moderator of effective coordination: "*the architecture is used as an **important coordination mechanism** to divide the work in the project*" (Baskerville & Pries-Heje, 2004, p. 247) (emphasis as in original document). Secondly, they tentatively related poorly coordinated activity to the quality of the product and the need for some minimal organisational, methodological, and/or software structure. They thought that structure may be needed more as an organisation and its products age and grow. The form this structure should take, however, is not yet clear within such organisations.

Confirmation of the findings of Baskerville et al. (2004) is provided by Harris, Aebischer, and Klaus's (2007) study of three small software product development companies in America. The three companies all used short iterative development (called micro-releases), with constant customer feedback, constant review sessions (prototyping), and a 'wait and see' approach to introducing new technologies when responding to "*the market's ebbs and flows*" (Harris, et al., 2007, p. 89).

Based on the assumption that successful commercial software development should exhibit agile characteristics, Hansson, Dittrich, Gustafsson and Zamak (2006) interviewed staff in five Swedish organisations to assess how closely they conformed to the four values published in the agile manifesto. They found some of the values were in evidence in some, but not all, of the organisations. In addition, within a single organisation individual projects were more or less compliant with those values. Projects that exhibited agile values were smaller and less critical. When the project team was small, cooperation and direct communication was used to coordinate and manage development, but as a project became larger (with a larger team) or more critical (e.g. life or financial criticality) more traditional and formal processes were applied. Shared code ownership⁵ was common but not universal, and documentation was handled in a variety of ways from formal to informal. Both customers and end-users were consulted frequently in all organisations, those that closely complied with the values and those that did not. The authors attributed this move to an agile state within these organisations as the application of common sense.

What these studies show is that some of the recommended practices of the agile development approach arise naturally when development teams need to cope with change while also aiming for fast delivery of software. This raises the question, what is an 'agile approach'? Is it a published agile method, or is it some subset of practices from an agile method? It cannot be a method that follows all of the practices in the manifesto, as no single published agile method achieves all of the manifesto principles (Conboy, Fitzgerald, & Golden, 2005). This confusion about the definition of agile software development has caused researchers to give the approach different names. Examples include:

- parallel or concurrent development (Baskerville, et al., 2001)
- adaptive development (Highsmith, 2000; Patel, 2003)
- incremental development (Paasivaara & Lassenius, 2004)
- Internet speed development (Baskerville & Pries-Heje, 2004)
- short cycle time development (Baskerville & Pries-Heje, 2004)
- flexible development (Agerfalk & Fitzgerald, 2006)
- incremental change (Rajlich, 2006)

⁵ When code is shared, or collectively owned, by all developers on a software project team this means any one of them can revise any parts of the code base, there is no one person who owns or has control over any part of the code (Williams, 2010, p. 15).

- short iteration development (Cao & Ramesh, 2007)
- organic development (Mathiassen & Pedersen, 2008; Xu & Ramesh, 2007)
- controlled-flexible development (Harris, Collins, & Hevner, 2009)

In summary, research indicates that an agile-like development approach arises naturally under certain environmental conditions, particularly time pressure. One of these studies emphasised the importance of architecture in coordinating this type of project. Further, there is a lack of clarity as to what ‘agile software development’ means and a number of alternative names for agile-like processes have emerged.

The next section reports on the common practice of method tailoring and the related effect of scaling agile methods to larger projects.

Agile Method Scaling and Tailoring

Method tailoring is reported in most studies of agile software development, particularly when scaling-up projects. The scalability of a method is its capacity to remain effective in situations that fall outside the boundaries of the environment for which it was designed. The tailoring or customisation of a method results when practices or techniques from the method are omitted, substituted, or additional techniques are grafted onto the method.

When first published, agile methods were considered difficult to scale up. They were designed for small, non-critical, green-field projects, with small collocated teams of 2-10 developers (Beck, 1999; Stapleton, 1997), and small projects or projects that can be broken down effectively into independent sub-projects suitable for small teams (Schwaber & Beedle, 2002). In addition, both Beck (2000) and Schwaber and Beedle (2002) stated that the whole set of practices in a published method must be used together to achieve the full benefits of an agile approach. The methods were considered less suitable, or at least untested, in large projects, mission critical projects, or projects involving legacy systems (Beck, 2000; Cockburn, 2002).

Reifer, Maurer, and Erdogmus (2003) identified problems of scaling likely to be problematic in the agile approach. This included the need for architecture definition, testing large interdependent systems, selecting the on-site customer from a large customer base, enabling communication between many small teams and geographically distributed teams, accommodating existing legacy systems and existing components, and the need for requirements engineering. Since then, large-scale projects have used agile methods, but scaling is not straightforward, and extensive method tailoring is

necessary (Cao, Mohan, Xu, & Ramesh, 2004; Elshamy & Elssamadisy, 2007; Lindvall et al., 2004).

XP can be successful with large teams and geographically distributed teams (Grossman, Bergin, Leip, Merrit, & Gotel, 2004; Schalliol, 2003; Woit, 2005), but the method must be tailored to the project environment. For example, a requirements management process was added to Woit's (2005) XP project in order to manage the requirements dependencies in a large complex project. Tractable and intractable problems with requirements engineering when using agile methods were found by Ramesh, Cao and Baskerville (2007) in a study of 16 USA organisations. Non-collocated teams are also viewed as problematic when XP is used, and support tools to enable effective communication, collaborative coding, project awareness, and coordination within such teams have been developed and used (Maurer, 2002; Reeves & Zhu, 2004; Schummer & Schummer, 2001).

Tailoring occurs to a greater or lesser extent in all software development projects whenever system development methods are employed, and it typically occurs because of situational contingencies peculiar to the project (Fitzgerald, 2000). Tailoring is normal practice when adopting agile methods. *"The ability to tailor any method is considered critically important given the complex and unique nature of each and every ISD environment, and in particular, one would logically expect that a method labeled as agile should be malleable"* (Conboy & Fitzgerald, 2007b, p. 218). Conboy and Fitzgerald (2007b) found tailoring was common after conducting a Delphi study involving 40 practitioners and academics with expertise in agile methods. Empirical reports of agile method tailoring are also very common (e.g. Aydin, et al., 2005; Broza, 2004; Fitzgerald, et al., 2006; Fruhling & de Vreede, 2006; Fruhling, Tyser, & de Vreede, 2005; Gunter, Gisler, & von Bredow, 2002; Nawrocki, Jasinski, Walter, & Wojciechowski, 2002; Xu & Ramesh, 2007).

For example, Fruhling and de Vreede (2006) used a two-year action research study to investigate how XP is used in practice when developing emergency response systems. The context was USA laboratories preparing for inter-organisational detection and handling of bio-security and terrorism threats. The developers used nine of the 12 XP practices, and added usability testing, code walkthroughs, and a technical writer to support the need for documentation.

Methods were both combined and tailored in the Intel Shannon case reported by Fitzgerald, Hartnet and Conboy (2006). In this interpretive exploratory case study, XP

and Scrum were combined, six XP practices were retained, some XP practices were eliminated, and some were substituted with similar practices from Scrum. This two-year study found that subsets of agile practices are interdependent, but it did not find that every XP or Scrum practice is dependent on all of the others, as the normative literature suggests (e.g. Beck, 2000; Schwaber & Beedle, 2002). They did find that many practices are linked because one practice influences and supports the use of another (e.g. pair-programming leads to shared code ownership; refactoring leads to a smaller code base, leading to a simpler design). Although synergistic effects of using a whole method (all of XP, or all of Scrum) have been proposed elsewhere, Fitzgerald et al.'s (2006) study is possibly the first to provide empirical evidence supporting the conjecture that subsets of practices are interdependent, and these subsets provide synergistic benefits within a whole project.

Method tailoring has implications for research into agile method use. Dyba, Moe, and Arisholm (2005) recognised this and said, *"it cannot be assumed that methodologies are followed or used consistently across all projects in an organization. Thus, it is not sufficient to ask if an organization has a methodology, as most surveys do, but rather, one must address the extent and nature of usage on individual projects"* (Dyba, et al., 2005, p. 449). They considered methodology usage to be a mediating variable in the relationship between system development methodology and organisational performance or project effectiveness.

In the previous section on spontaneously occurring agile methods, this review found that agile processes can occur spontaneously under certain conditions, and this makes studying the 'agile approach' difficult. The method tailoring described in this current section compounds the problem, because what an agile approach consists of may differ depending on project contingencies. The following section addresses research that attempts to circumvent this issue by developing a generic definition of agility in the context of software development.

Agility

Agility is considered an important quality in organisations and in organisation research (Dove, 2005; Lyytinen & Rose, 2004). IT executives rank 'speed and agility' as the 8th most important critical success factor within their organisations (Luftman & McLean, 2004, p. 93). But the concept of agility is not well defined. Within agile and software development literature agility is also called flexibility, adaptability, adaptivity, leanness, or organic development (Agerfalk & Fitzgerald, 2006; Harris, Hevner, & Collins, 2006;

Highsmith, 2000; Little, 2005; MacCormack, et al., 2001; Mathiassen & Pedersen, 2008; Meso & Jain, 2006; Salo et al., 2004). Leanness and flexibility are terms from manufacturing which can be traced to the 1950s (Conboy & Fitzgerald, 2007a) and the Toyota Production System (Ohno, 1988), and early agile approaches to software production (Aoyama, 1996, 1998).

Conboy (2004) began a study of agility when he found his work mapping agile methods to the agile manifesto showed no agile method conforms exactly to the agile manifesto principles. He initially formulated the concept of agility based on a comprehensive review of agility in management, organisational behaviour, and manufacturing literature. The definition of agility emerged as follows: “...*the continual readiness of an entity to rapidly or inherently create change, proactively or reactively embrace change, and learn from change, through customer value-adding components and relationships with its environment*” (Conboy & Fitzgerald, 2007a, p. 209). The three main concepts in this definition are agile drivers, agile capabilities, and agile value (see Figure 3 for an explanatory illustration). Agile drivers are the types of change that may impact a project, and include internal, immediate environment, and general environment change. Agile capabilities incorporate concepts of creation, pro-action, reaction, and learning; capabilities the selected method must provide. Agile value is related to the business value the method or partial method will provide to the business customer.

Agile Drivers, Capabilities, and Value

Figure 3 Agile drivers, capabilities and value from Conboy and Fitzgerald (2007a)

The advantage of this framework is it enables the decoupling of studies of an agile approach from the study of ‘methodology’, that is, the use of any one particular agile method. Another advantage is that it “*dispel[s] the notion that an activity can be labeled as completely agile or non-agile. It depends on the context in which it is used*” (ibid, p. 213). This framework provides a comprehensive means to assess the agility of a project and to assist in method selection. Each concept is carefully drawn from literature on information systems development, the agile approach, lean manufacturing, amethodical development, and method engineering. The assessment of agility remains a qualitative exercise even with this framework, because the agility concept has not been operationalised to the extent that it can be measured.

Agility in system development is linked with organisational agility in a major study by Lyytinen and Rose (2005) in which they outlined a theory of software development agility. This study was an extension of that reviewed previously in section 2.1.3 Agile Methods, regarding IT innovation (Lyytinen & Rose, 2003b). They based their ideas on Swanson’s (1994) model of IT innovation, and March’s (1991) exploration-exploitation dichotomy. Lyytinen and Rose’s (2005) model links vendors and manufacturers (who

make type 0 innovations, such as new operating systems), IS development organisations (who take up type I innovations, such as new development processes), and IS deploying organisations (who take up type II innovations, such as new IT solutions). They then investigated “*the extent to which changes in Type 0 innovation can lead to innovations in Type I, such as agile development, and the consequent fast absorption of type II innovations (business agility)*” (Lyytinen & Rose, 2005, p. 206). This ability to take up innovations they ascribe to an organisation’s explorative and exploitative capabilities. They tested their model in a multi-case study of seven organisations that adopted Internet computing over a 5-year period. The study, using data from three USA and four Finnish web development organisations included interviews with 17 managers regarding changes in their software development practices. Interviews covered three time points during the period from 2000-2004. Their inductive data analysis showed the use of formal methodologies was related to the phase the software development company was in – explorative or exploitative. Exploration phases, when product innovation is intense, tend to eschew methodology, and when exploitative phases emerge and incremental product improvement becomes the focus, the concern for quality rises and methodologies tend to reappear.

This section has shown that a theoretical concept of agility in the software development context has been defined, and that a relationship has been shown to exist between software development agility and organisational agility. The following section investigates research that uses theory in studies of agile software development.

Theory and Agile Methods

The agile development approach is seldom discussed or investigated from the perspective of existing theory, nor have many new theories been developed to explain the approach or its effectiveness. In 2005, Conboy summarised the state of agile development research: “*The various agile methods in existence lack sufficient grounding in management theory, organisational theory, and indeed theory behind all the fields and disciplines which comprise ISD*” (Conboy, et al., 2005, p. 36). Nerur and Balijepally (2007) proposed theories they considered applicable to agile development including Action Learning Theory, Dewy’s pragmatism, Ashby’s law of requisite variety, and the theoretical principles of holographic organisations. These authors consider the agile approach a “*new epistemology of software development*” (Nerur & Balijepally, 2007, p. 82) which they likened to similar changes in the fields of strategic management and building design. Dingsoyr, Dyba and Abrahamsson (2008) proposed a different list of

potentially relevant theories: task-technology fit, theory of planned behaviour, contingency theory, complexity theory, social learning theory, social network theory, socio-technical theory, organisational learning theory, and the knowledge-based theory of the firm. These theories have not yet been applied in the field to explore and explain agile software development.

These discussions on theorising agile approaches make it clear there are different views on the use of theory in this context. The first is that agile approaches can be studied using existing theory. Another possibility is that the agile approach can be used to extend existing theory. A third view is that theory in the field of system development –including agile system development - should be built from observed practice (Fitzgerald, 2000; Karleinz Kautz & Zumpe, 2008).

Taking the first view that the agile approach can be studied using existing theory, Cao and Ramesh (2007) discussed the relationship between agile software development and three organisation theories. First they noted that “*no unified theory exists that can comprehensively explain these practices and policies*” (Cao & Ramesh, 2007, p. 42). Then they compared the principles of agile methods with dynamic capabilities theory (Eisenhardt & Martin, 2000), coordination theory (Van de Ven, et al., 1976), and double loop learning theory (Argyris & Schon, 1974). They concluded that each of these theories explains some aspects of the agile development approach, but not all (Van de Ven’s (1976) coordination theory and its applicability to agile development is discussed in a subsequent section of this review).

Mindfulness is a theory applied to agile software development by Butler and Gray (2006) in a conceptual essay, and used by Matook and Kautz (2008) in a single interpretive case study. Mindfulness can be individual or collective (involving a group), and is defined as “*a way of working characterized by a focus on the present, attention to operational detail, willingness to consider alternative perspectives, and an interest in investigating and understanding failures*” (Butler & Gray, 2006, p. 2). Matook and Kautz (2008) proposed that mindfulness explains the effectiveness of the agile approach.

Complex adaptive systems (CAS) theory has also been proposed as a lens for studying agile software development. A CAS is a system “*capable of adapting to its external environment and its internal state so that it survives despite the circumstances that befall it*” (Meso & Jain, 2006, p. 20). In CAS theory, a complex adaptive system is self-organising, emergent, and more than the sum of its parts. These authors’ mapped CAS principles to common agile practices and then drew on empirical data to illustrate how

the theory explains the agile approach. Although some of their arguments are reasonable, this theory only addresses the social and organisational aspects of software development; its applicability to the technical artefacts of development, such as the impact of development tools, software products, architectural frameworks, and the requirements linking these artefacts, is questionable. Since agile software development is closely entwined with technology (particularly XP, which mandates automated testing), this theoretical idea is one-sided, providing only a social/organisational perspective on agile software development.

The knowledge management theory of Tuomi (1999) was applied by Kahkonen and Abrahamsson (2003) to explain how XP practices contribute to knowledge creation in an organisation. They concluded that more theoretical development is needed to explain agile approaches, and that Tuomi's management theory can only explain how some XP practices contribute to knowledge creation.

An extension to control theory to accommodate the agile approach was proposed by Harris, Collins, and Hevner (2009). Control describes mechanisms that ensure actors contribute to common organisational goals (Kirsch, 1996). Harris, et al. (2009) based their work on that of Kirsch (1996, 1997) and Ouchi (1977, 1979, 1980) who formulated the concepts of behavioural, outcome, self, and clan control. Harris, et al. (2009) applied control theory to case study data from seven software development projects using a range of system develop methodologies, traditional (plan-driven), controlled-flexible (agile), and ad hoc (no methodology). Based on empirical data they identified a new form of control in the controlled-flexible projects named 'emergent outcome control', which is suitable for guiding development under conditions of increased time pressure and uncertainty.

The third view of theory and the agile approach is that theory in the field of software development should be built from observed practice (Baskerville & Pries-Heje, 2004; Curtis, et al., 1988; Fitzgerald, 2000; Karleinz Kautz & Zumpe, 2008; Xu & Ramesh, 2007). Such research typically takes a grounded theory, ethnographic, or phenomenological approach to develop explanatory concepts. As the earlier sections of this review of the agile approach have shown, this research has contributed significantly to understanding the practice of agile software development. It has also provided evidence that agile software development is a new paradigm in software development that has emerged to meet the need for organisations to cope with a faster

paced-business environment, and the technological environment of development for the Internet.

This section has shown that, while researchers have called for the application of existing theory to explain agile software development, there have been few attempts to do this, and even fewer attempts to extend existing theory to accommodate the agile approach. What is largely missing in the literature is new theory building to explain the effectiveness of the agile software development approach.

Summary of Agile Methods Research

Agile software development is a unique approach to information systems development. In the years since the individual agile methods were published, research has matured from anecdotal reports, to descriptive studies, to the application and extension of existing theory studies based on rigorous research. There has been very little theory building, and almost no theory testing, although it is just now beginning to appear (e.g. Cummings, Espinosa, & Pickering, 2009).

One problem is a lack of clarity about what constitutes agile software development. Agile software development can be adoption of an agile method, using one or more agile practices, using short iterations, subscribing to the principles and values of the agile manifesto, or developing emergent agile practices under pressure. All of these scenarios have been portrayed as agile software development by various authors.

Overall, there has been a lack of theoretical development to explain the agile software development approach. There are some exceptions, for example Baskerville and Pries-Heje (2004), Lyytinen and Rose (2005), and Harris, Collins, and Hevner (2009) have produced theory building work based on substantial empirical studies. Given an agile approach impacts individuals, projects, and organisations there is much existing theory in those realms that could be applied or extended to explain how and why agile software development is effective.

This review of the agile software development literature has found some information on coordination in this context. Empirical studies have identified the following:

- Coordination activities - actions taken to achieve coordination (e.g. pair programming, meetings, refactoring, automated builds)
- Coordination artifacts - non-human things that contribute to coordination such as documents, models, or tools (e.g. the wall, software architecture)
- Spatial arrangements - geographical aspects of coordination (e.g. co-location)

- Temporal coordination - the time element of coordination (e.g. iterative development)
- Cognitive factors - dealing with perception, information, and understanding (e.g. mindfulness)

This concludes the section on the agile development approach. Coordination is the topic of the next major section. Once coordination is explained and its literature examined, the review considers studies focusing on coordination and agile software development.

2.2 COORDINATION

Coordination is a theme in the research on organisations, IS projects, and teamwork and each of these areas is pertinent to agile software development. Therefore, this section reviewing coordination literature is organised around these three themes.

Another important contributor to coordination research is an interdisciplinary 'Coordination Theory'. This theory is the foundation of the initial conceptual framework of this research, so it is described in detail.

This section of the literature review is organised as follows. First, the review addresses coordination in organisation studies, followed by a description of Coordination Theory. Then the literature on coordination in IS projects, teamwork studies, and agile software development is reviewed. Perspectives on coordination arising in this body of literature are then summarised. This section concludes with a review of research linking coordination with IS and IT project success.

Before proceeding, it is useful to know how coordination is defined. Table 2 provides a selection of definitions from various fields. The commonalities in these definitions include the ideas that coordination involves more than a single person, their activities are interdependent, and they are working towards a common goal.

Table 2 Selected definitions of coordination

Definition	Research Field
<i>"Work or act together effectively."</i> (Allen, 1990, p. 253)	Dictionary definition
<i>"The integration or linking together of different parts of an organization to accomplish a collective set of tasks."</i> (Van de Ven, et al., 1976, p. 358)	Organisation studies
<i>"(1) people work collectively; (2) the work is interdependent; and (3) a goal, task, or piece of work is achieved."</i> (Okhuysen & Bechky, 2009, p. 469)	Organisation studies
<i>"When multiple actors pursue goals together, they have to do things to organize themselves that a single actor pursuing the same goals would not have to do. We call these extra organizing activities coordination."</i> (Malone, 1988, p. 5)	Interdisciplinary
<i>"Coordination is the managing of dependencies between activities."</i> (Malone & Crowston, 1994, p. 90)	Interdisciplinary
<i>"Coordination means the spatial and temporal synchronization of overt behaviors of two or more people so that those actions fit together into an intended spatial and temporal pattern."</i> (Arrow, McGrath, & Berdahl, 2000, p. 42)	Teamwork studies
<i>"Coordination is then perceived as a problem of sharing, integrating, creating, transforming, and transferring knowledge."</i> (Kotlarsky, van Fenema, & Willcocks, 2008, p. 96)	Knowledge management

For the purposes of this study, Malone's (1988) definition of coordination is a useful starting point, since a software development project team involves multiple actors working together to pursue the common goal of developing a software product. Later, the more precise definition provided by Malone and Crowston (1994) is used in developing the conceptual framework for this study, and for analysing coordination in selected agile software development projects.

2.2.1 COORDINATION IN ORGANISATION STUDIES

Coordination is a pervasive topic in organisation studies. Okhuysen and Bechky (2009) provide a broad review of coordination within organisations. Their review moves from early coordination studies focusing on the influence of coordination on organisation design and organisational efficiency, to more recent approaches focusing on how coordinated work is achieved, or emerges, at the level of work practice.

In early organisation studies, coordination was considered a key determinant of organisation structure (Galbraith, 1973, 1977; Malone, 1987; March & Simon, 1958; Mintzberg, 1980; Thompson, 1967; Van de Ven, et al., 1976). The traditional information-processing view of organisation structure was formulated by Galbraith (1973, 1977), and is based on three ideas: contingency, coordination, and uncertainty. The most effective organisational structure fits the environmental contingencies an organisation faces. Organisation structure is determined by the way in which tasks for achieving organisational goals are divided among actors, and how those tasks and actors are then coordinated to form a cohesive whole. Furthermore, the uncertainty an organisation faces is a function of the gap between the information an organisation needs to carry out a task, and the information available to the organisation.

Coordination at the organisational level has various modes and sub-modes.

Coordination by programming, also called impersonal mode, occurs when “*a codified blueprint of action is impersonally specified*” (Van de Ven, et al., 1976, p. 323).

Impersonal mode coordination 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 characterises this coordination mode.

Coordination by feedback (March & Simon, 1958), also called coordination by mutual adjustment (Thompson, 1967), can take place in either group mode or personal mode (Van de Ven, et al., 1976). Group mode coordination is achieved with scheduled or unscheduled meetings. Personal mode coordination can be vertical or horizontal. Vertical coordination involves communication via supervisors and line managers whereas horizontal coordination occurs through one-to-one communication between actors in a non-hierarchical relationship, although a designated coordinator with no formal authority may be present (Lawrence & Lorsch, 1967).

Based on earlier theoretical work, Van de Ven et al. (1976) identified task uncertainty, task interdependence, and size of work unit, as fundamental determinants of coordination mode. In a key empirical study, they investigated the impact of these three factors on impersonal, personal, and group coordination modes within organisational subunits in a large American employment security agency. Their quantitative study with 1077 respondents, supported with qualitative data, showed:

As task uncertainty increases:

- Coordination by horizontal communication channels and group meetings (mutual adjustment) increases significantly. These mechanisms substitute for vertical hierarchical coordination and impersonal coordination.
- Impersonal coordination decreases significantly.
- The use of vertical communication channels is invariant.

As task interdependence increases:

- Impersonal and personal coordination mechanisms remain invariant.
- Group coordination increases significantly (especially scheduled meetings).
- There is a greater use of all coordination mechanisms combined.

As work unit size increases:

- The use of hierarchy increases.
- The use of horizontal channels and group meetings remains invariant.

Coordination cost is another concept introduced by Van de Ven et al. (1976). This cost is associated with the amount of interpersonal communication between work unit members. This cost is lower when impersonal coordination is used, and higher when horizontal channels and group meetings are used.

Workflow categories are also defined by Van de Ven et al. (1976), based on Thompson's (1967) work. Workflow can be independent, sequential, reciprocal, or team. In independent workflow, work enters a work unit and actors perform work activities independently, work does not flow between them. Sequential workflow occurs when work enters a unit and passes between actors in a single direction; it then passes out of the work unit. In the case of reciprocal workflow, work enters the work unit and passes back and forth between actors. In team workflow, work enters the work unit and the actors diagnose, problem-solve, and collaborate as a group working concurrently to deal with the work. Van de Ven et al. (1976) use these workflow variables to investigate workflow at the organisational subunit level. They found increases in workflow interdependence from independent, to sequential, to reciprocal, to team arrangements is associated with increasing group coordination mechanisms.

Further theoretical work mapping organisation structure to coordination mechanisms was carried out by Mintzberg (1980) who defined five pure types of organisation structure and their dominating coordination mechanism. Simple organisational structures are coordinated by direct supervision, machine bureaucracies are coordinated by standardisation of work, professional bureaucracies by standardisation

of skills, and the divisional form by standardisation of outputs. The final form, adhocracy, is characterised by mutual adjustment whereby individuals coordinate their own work using informal communication. Adhocracies occur in dynamic and complex environments, youthful organisations, those with organic structures, and organisations involved in sophisticated innovation. Adhocracies consist of multidisciplinary teams that treat problems as unique challenges requiring creative solutions, and work on projects that combine planning, design, and execution work.

Certain agile practices seem to reflect some of these coordination concepts, in particular the idea of mutual adjustment. For example, in the Scrum method (Schwaber & Beedle, 2002) group mode coordination, which relies on scheduled and unscheduled meetings, 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, 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., 2001a), since team workflow involves a group working concurrently to diagnose, collaborate, and problem solve to process the work. Finally, an agile approach is likely to engender high coordination cost (Van de Ven, et al., 1976) due to its reliance on mutual adjustment and group mechanisms. Van de Ven et al.'s (1976) empirical finding that as unit size increases impersonal modes increase, but horizontal-personal and group modes are invariant (Van de Ven, et al., 1976) is contradicted in the agile approach. In XP, Scrum, DSDM, Crystal, and ASD, a team size (assuming team size is equivalent to unit size) of 2-10 is considered optimal (Strode, 2007), and if team size increases beyond this maximum, problems are expected to occur that cannot be resolved with agile practices (Beck, 1999; Lindvall, et al., 2004; Schwaber & Beedle, 2002).

Two issues arise when applying organisation coordination theory to agile software development. Organisation-level coordination research is based on theoretical and empirical work carried out in the pre-1990 era, and the IS organisational environment of 2012 may be very different. Therefore, it may not be appropriate to apply these theoretical concepts in a study of agile software development, since agile methods have been in common use only in the last decade. A second issue with applying these coordination concepts to explain and predict the agile approach is the assumption that a project is the same as an organisation. While some researchers have made this assumption (e.g. Barki, Rivard, & Talbot, 2001; Nidumolu, 1995; Xia & Lee, 2005) it may

not be correct. Organisation level theories may not hold at the project level due to the effect of time constraints that might not be present in permanent organisations (Barki, et al., 2001).

In summary, there are a number of coordination modes specified in organisation theory; agile development principles and practices seem to reflect some, but not all, of them. The concept of coordination therefore provides a link between organisation theory and agile development approaches.

Many concepts from organisation theory mentioned in this section are used in the coordination studies of IS projects reviewed in the following sections. Before reviewing literature on coordination in IS projects, teamwork, and agile software development, a major theory of coordination is described first.

2.2.2 COORDINATION THEORY

One theory focuses exclusively on coordination. This interdisciplinary theory of coordination is the basis for the conceptual framework guiding this study, and is also used in the data analysis phase. Therefore, this section describes the theory in detail. In this thesis, the theory is called 'Coordination Theory' to distinguish it from other theories with similar names.

The History of Coordination Theory

In 1988, Malone proposed a multi-disciplinary theory of coordination by drawing together ideas and theories about coordination from the fields of organisation theory, management science, psychology, computer science, and economics (Malone, 1988). He envisioned three practical uses for the theory: designing automated tools to support cooperative work, designing multiple parallel computer processors, and designing flexible organisations and work processes. He thought a multi-disciplinary theory would provide ways to study the coordination of groups of people, groups of computer processors, interacting program modules, and hybrid groups of people and computers.

Malone and others set up the 'Center for Coordination Science' at MIT Sloan School of Management in the late 1980s to study coordination in its various forms (Malone, 1989). Working with Malone, Crowston began investigating computer-supported cooperative work, and they developed the core concepts of Coordination Theory (Malone & Crowston, 1990). Next they developed the Process Handbook, which is a repository of business process knowledge organised using concepts from Coordination

Theory and object-oriented software modelling (Crowston & Osborn, 2003; The MIT Process Handbook Project., 2003).

In 1994, a survey of the work on Coordination Theory was undertaken summarising its core theoretical concepts, and discussing methods for analysing coordination in different types of systems. This article also provided examples of the application of Coordination Theory in organisational structure, cooperative work tools, and distributed and parallel computer systems (Malone & Crowston, 1994). The article outlined a research agenda for the study of coordination, including the need for methods to represent and classify coordination processes, methods for analysing coordination cost, the need for empirical studies of human systems, and the need for formal models of coordination processes. The article also reviewed coordination studies in computer science, economics, operations research, organisation theory, and social and biological systems.

In 2006, Crowston, Rubleske, & Howison (2006) published a ten-year retrospective of Coordination Theory. They discussed the state of the theory, and provided an impact analysis showing that the theory, as defined in Malone (1988), and Malone and Crowston (1990, 1991, 1994), had had a moderate impact, with 287 references from 1989 to 2004 (compared with the high-impact Technology Acceptance Model (Davis, 1989) which has about 800). They found applications of Coordination Theory range across the organisational domains of business processes, supply chains, and organisational simulations, and within the computer science domains of software engineering and systems design. This theory has maintained its utility. As at 15 February 2012, the article by Malone and Crowston (1994) in the Web of Science has 617 citations, and the Scopus database shows 732 citations.

Coordination Theory Concepts

Malone and Crowston's (1994) theory provides a way to study and improve coordination processes. The precise definition underlying coordination theory is: *"Coordination is managing dependencies between activities."* (Malone & Crowston, 1994, p. 90). This is elaborated by Crowston and Osborn who explain, *"actors performing activities face **coordination problems** arising from dependencies that constrain how the activities can be performed. These coordination problems are managed by activities that implement **coordination methods**"* (Crowston & Osborn, 2003, p. 337).

In Coordination Theory, coordination encompasses collaboration, cooperation, and competition. Communication is considered a special case of coordination for managing

a producer-consumer dependency by the transfer of information (Malone & Crowston, 1994).

The principle concepts in Coordination Theory are dependency and coordination mechanism. A dependency occurs when the progress of one action relies upon the timely output of a previous action, or the presence of some specific thing. Dependencies lead to potential or actual constraints on projects. Coordination is achieved through coordination mechanisms (e.g. queuing at a supermarket, allocating component development to different software developers) that address dependencies in a situation. The theory is supported by a high-level taxonomy of generic dependencies and examples of possible coordination mechanisms for managing those dependencies developed from organisation theory, economics, and computer science literature. The taxonomy is reproduced in Table 3.

Table 3 Dependencies and coordination mechanisms (Malone & Crowston, 1994)

Dependency	Coordination mechanism
Shared resources [sharing] <ul style="list-style-type: none"> • Task assignments 	'First come – first serve', priority order, budgets, managerial decision, market-like bidding (same as for shared resources)
Product/consumer relationships [flow] <ul style="list-style-type: none"> • Prerequisite constraints • Transfer • Usability <ul style="list-style-type: none"> – Design for manufacturability 	Notification, sequencing, tracking Inventory management (e.g. Just In Time, Economic Order Quantity) Standardization, ask users, participatory design Concurrent engineering
Simultaneity constraints [fit]	Scheduling, synchronization
Task/subtask	Goal selection, task decomposition

Dependencies are central to Malone and Crowston's (1994) Coordination Theory because, "*if there is no interdependence, there is nothing to coordinate*" (Malone & Crowston, 1994, p. 90). Note that, as an interdependency can be decomposed into two distinct dependencies (i.e. if A and B are interdependent, then A depends on B and B depends on A), later work on this theory used the term dependency in preference to interdependency (Crowston, 2003; Crowston, et al., 2006; Malone et al., 1999).

Coordination Theory focuses on dependencies between activities rather than dependencies between actors. Crowston (1994) found that organisation studies of coordination focus on dependencies between actors whereas artificial intelligence studies of coordination focus on dependencies between activities. Crowston (1994) was interested in using the ideas from artificial intelligence to inform the theory so he

focused on linkage patterns between activities as a way to define dependencies. Actors still play a part, but they are conceptualised as a form of resource.

Before proceeding, the following definitions are useful in understanding Coordination Theory:

- An actor is a thing that executes the process of interest. An actor may be human or non-human, an individual, a collective or an organisational subunit, depending on the level of analysis (Malone & Crowston, 1990).
- A resource is any thing created or used by activities. Resources include both material objects and the effort of actors. Actors are also a type of resource (Crowston, 2003).
- A goal is a desired state of the world (Crowston, 2003). Goals can be decomposed into sub goals.
- A task is performed by carrying out activities to achieve a goal. A task can be decomposed into sub-tasks (Crowston, 2003). Tasks can be viewed at different levels of analysis, for example 'performing a build' in software can be considered a single task at one level, but can be viewed as a number of sub-tasks at another level, for example, 'locate components', 'integrate components', and 'run build application'. Coordination Theory does not distinguish precisely between task and activity.

Malone, et al. (1999) proposed three elementary types of dependency between activities and resources: fit, flow, and sharing. Furthermore, his idea was that all dependencies between resources and multiple activities can be analysed as specialisations or combinations of one of these elementary types. His three types are defined thus:

- A flow dependency occurs when one activity produces a resource used by another activity
- A sharing dependency occurs when two or more activities use a single resource
- A fit dependency occurs when multiple activities produce a single resource

Crowston (2003) decomposed dependencies further. First, he amalgamated the components of coordination - actors, tasks, activities, goals, and resources - into two categories: tasks and resources. Tasks include activities and goals, and resources include actors, material goods, or the effort of actors. With this simplification,

coordination becomes a mechanism used to manage the problems created by the dependencies between task and resource, task and task, and resource and resource.

The three elementary task-resource dependencies along with other possible dependencies as defined by Crowston (2003) are shown in Figure 4.

Figure 4 Dependency relationships adapted from Crowston (2003)

These dependencies are described as:

- *Single-resource multiple-task dependency* – cases (1), (2), and (3) all require coordination. Case (1) is sharing, requiring coordination to share a resource. Resources may be consumable or reusable, shareable or non-shareable, which determines how they may be coordinated. Case (2) is flow, requiring coordination because the resource is produced by one task and used by

another. Tasks are often performed in a particular sequence requiring coordination. Three types of flow are:

- a) usability, whereby the resource must be in a specific form for it to be usable,
- b) prerequisite, whereby the resource must be produced in a final form before it can be consumed, and
- c) accessibility, whereby the resource may need to be in the correct location before it can be used.

Case (3) is fit, requiring coordination because multiple tasks produce a single resource. The resource may be in parts so the constraint would then be to ensure the parts fit together.

- *Single-task multiple-resource dependency* – cases (4), (5), (6). Case (4) and (5) are not likely to require coordination. Case (6), task uses multiple resources, may require coordination. This is because resources may need to be scheduled, or their availability synchronised in some way.
- *Single-task single-resource dependency* - cases (7) and (8). Task produces resource (8) requires no coordination. Task uses resource (7) does require coordination and can be decomposed further based on the coordination problem of acquiring the necessary resource. Coordination may involve identifying needed resources, identifying available resources, gathering information about those resources, selecting the resource, and allocating the resource.

Further forms of dependency are task-task dependencies and resource-resource dependencies, which involve simultaneity, composition, and integration dependency subtypes. A simultaneity dependency occurs when tasks must be performed simultaneously. Composition dependencies occur when tasks are decomposed into subtasks. Integration is then needed to recompose the subtasks. In a resource-resource dependency, composition and integration dependencies occur. Crowston (2003) notes that these cases can often be reconceived as one of the cases in (3) above.

Dependency is the first major concept of coordination. The second is 'coordination mechanism'. Coordination Theory is founded on the idea that dependencies in a situation are managed by coordination mechanisms. This idea led Malone et al. (1999) to propose the following:

- There are generic dependencies common to many different situations

- Each dependency may be managed by any number of alternative coordination mechanisms
- Coordination mechanisms may be ubiquitous, that is the same coordination mechanism may be found in many different processes
- It is feasible to swap one coordination mechanism with another to address the same dependency in a situation

The idea of swapping coordination mechanisms is illustrated in Malone and Crowston's (1994) taxonomy of generic dependencies and coordination mechanisms (see Table 3). The Process Handbook, developed from Coordination Theory, is a source of numerous examples of potentially interchangeable coordination mechanisms, some unique to different lines of business, including mechanisms for system development (Herman & Malone, 2003; The MIT Process Handbook Project, 2003). In addition, McChesney and Gallagher (2004) have elaborated Malone and Crowston's (1994) dependency and coordination taxonomy with examples from software engineering projects, and Crowston and Kammerer (1998) with examples from the system requirements process.

The idea of substituting coordination mechanisms is important in Coordination Theory. Where there are multiple possible coordination mechanisms to manage a dependency, then it may be possible to substitute one coordination mechanism for another. This gives a way to redesign and improve processes, because by understanding which substitutions are possible, organisations can improve the flexibility of their business processes. Since coordination processes determine organisational form (March & Sutton, 1997), this theory offers a way to redesign organisations to achieve "*more flexible and more satisfying ways of organizing collective human activity*" (Malone & Crowston, 1994, p. 111).

Research using Coordination Theory

Coordination Theory has been used to investigate various aspects of system development. This includes the software engineering process (Crowston & Kammerer, 1998), the requirements development process (McChesney & Gallagher, 2004), developer task-assignment in FLOSS⁶ projects (Crowston, Li, Wei, Eseryel, & Howison, 2007), and to analyse communication within agile development projects (Pikkarainen, Haikara, Salo, Abrahamsson, & Still, 2008) (reviewed in a later section on Coordination in Agile Software Development).

⁶ Free/Libre Open Source Software

The Status of Coordination Theory as a ‘Theory’

The status of coordination theory has changed since its introduction in 1994. When Malone and Crowston (1994) first described Coordination Theory they said “*We use the term theory with some hesitation because it connotes to some people a degree of rigor and coherence that is not yet present in this field*” (Malone & Crowston, 1994, p. 88). Later, Crowston, Rubleske, and Howison (2006) began their 10-year retrospective by discussing the status of Coordination Theory as a theory, and defined it as a pattern model, following Kaplan (1998), because it explains phenomena by showing how they fit a known pattern. The retrospective discussed the success of Coordination Theory as measured against the criteria of applicability, salience, testability, multiple uses for the theory, fit with existing theory, understand-ability, and ease of use, public availability, and communication in a variety of settings. Coordination Theory meets each of these criteria except testability because, although concepts are defined in Coordination Theory, there are no propositions or hypothesised relationships. Crowston, Rubleske, and Howison (2006) acknowledge this when they state, “*challenges for future research include developing testable hypotheses (e.g. About the generality of coordination mechanisms)*” (Crowston, et al., 2006, p. 135).

Coordination Theory is a ‘theory for analysis’ according to Gregor’s (2006) theory-classification scheme. This type of theory is used for describing and analysing a situation. Theories of this type include “*classification schema, frameworks, or taxonomies*” (Gregor, 2006, p. 623). Confusingly, Coordination Theory is called both a taxonomy and a typology by Crowston (2003). Bacharach (1989) argues that taxonomies and typologies are not theories, whereas Doty and Glick (1994), like Gregor (2006), believe they are theories. They define taxonomy as a scheme based on discrete decision rules for categorising phenomena where each category in the taxonomy is mutually exclusive, and the overall taxonomy is collectively exhaustive. A typology is a set of ideal types where each type consists of a unique combination of attributes and each type has a causal relationship to some relevant outcomes. According to this definition, Coordination Theory could be considered a taxonomy rather than a typology.

Although Coordination Theory is not a theory for prediction, but rather for analysis of a situation, its central tenet that coordination mechanisms exist to address dependencies, has an important role in this research study. Coordination theory underpins the initial conceptual framework and provides a means to analyse the research data.

Coordination research in studies of IS projects draws on organisation-level coordination concepts and Coordination Theory. The following section of the review focuses on this research.

2.2.3 COORDINATION IN IS PROJECT STUDIES

Coordination is a consistent theme in IS project studies because coordination is critical to effective software development (Amrit & van Hillegersberg, 2008; Curtis, et al., 1988; Grinter, 1996; Herbsleb & Mockus, 2003; Kraut & Streeter, 1995; McChesney & Gallagher, 2004; Nidumolu, 1995; Ovaska, Rossi, & Marttiin, 2003; Toffolon & Dakhli, 2000). Well-coordinated development is assumed to not only produce software faster, but also to produce software of higher quality and at lower cost (Espinosa, et al., 2007). This section focuses first on research on coordination in software projects in general, followed by distributed software projects.

Curtis et al. (1988) were the first to identify the negative impact of poor coordination on software development projects when they carried out a seminal field study into the problems of large-scale software design and development. After interviewing staff on 17 projects within nine companies, they identified three major and interdependent problems:

- A lack of application domain knowledge among development teams
- Fluctuating and conflicting requirements
- Communication and coordination breakdowns

Coordination breakdowns affected each organisational level: the business milieu, the company, the project, the team, and the individual.

Building on Curtis et al.'s (1988) findings, Kraut and Streeter (1995) argued that project size and project complexity increase the difficulty of coordinating software development efforts. They measured coordination success in 65 projects with 563 respondents using a survey within one large software development company. The company was using a waterfall development process (described in section 2.1.1. Historical Perspective). They found that more technically certain projects, and "*projects that were older, smaller, and less interorganizationally interdependent were better coordinated*" (Kraut & Streeter, 1995, p. 79). They also found, as predicted by organisation theory (Van de Ven, et al., 1976), that when projects are uncertain interpersonal networks are more beneficial; when projects are large they need a mixture of formal and informal communication to be effectively coordinated; and that

personal communication is important for successful coordination. They also found *“staff members’ assessment of their project’s coordination strategy correlates with customers’ satisfaction with the software development company and the software it produces”* (Kraut & Streeter, 1995, p. 77).

The relationship between coordination in software development projects and project effectiveness is investigated in two quantitative studies using the information processing view from organisation theory (Galbraith, 1977; Van de Ven, et al., 1976). Andres and Zmud (2001) used a laboratory experiment with 80 students working on a scaled-down programming task to investigate the impact of requirements’ uncertainty, and vertical and horizontal coordination on software project performance risk. They found an organic coordination strategy (informal, cooperative, and decentralised) enhances team productivity, and task interdependence significantly reduces productivity. Nidumolu (1996) investigated horizontal coordination and vertical coordination in software projects and found they play different roles. In his study of 64 projects, under conditions of high uncertainty, horizontal coordination enhanced software flexibility in two ways: by improving team communication, and by increasing the range of product design alternatives considered. Vertical coordination was found to enhance process control.

Research in software engineering takes a very different view of coordination. In that field, coordination is considered to be critically related to system architecture. The architecture of a system is the way that software components are organised in a software development environment (Kruchten, 2008). Curtis et al. (1988) acknowledged the importance of architecture knowledge in successful system development, and Baskerville and Pries-Heje (2004) found it to be an important moderator of coordination in Internet development projects. Normative agile methods literature (listed in Table 1) ignores system architecture (Lindvall, et al., 2004; Nord & Tomayko, 2006). Although the first publication explaining XP provided a ‘system metaphor’ practice designed to convey the essence of the system architecture to the team (Beck, 2000), this practice is seldom used in projects (Conboy, et al., 2005). This lack of attention to system architecture in agile methods has been recognised as a potential problem that could lead to suboptimal design decisions (Dyba & Dingsoyr, 2008, p. 836).

Software engineers have proposed a reciprocal relationship between the architecture of a system and the structure of the organisation creating the architecture (Conway,

1968; Herbsleb & Grinter, 1999; Ovaska, et al., 2003). ‘Conway’s law’ is a statement about this relationship based on the observation that *“two software modules A and B cannot interface correctly with each other unless the designer and implementer of A communicates with the designer and implementer of B. Thus the interface structure of a software system NECESSARILY will show a congruence with the social structure of the organisation that produced it”* (Conway, 1968, p. 1). Based on this idea, Conway argued that it is important for organisations to be both lean and flexible in their communication and organisational structures so that the teams creating the software are readily able to adjust and improve the system architecture. This ‘law’ has implications for both co-located and globally distributed software development as it implies a need for architectural and organisational fit (Herbsleb, 2007). Herbsleb (2007) discussed coordination of globally distributed software development and noted that understanding the relationship between software dependencies and task dependencies would enable project managers to predict the need to co-locate teams. However, although a relationship between architecture and organisational structure was confirmed by MacCormack, Rusnak, and Baldwin (2008), no clear directional relationship has been demonstrated (Amrit & van Hillegersberg, 2008; Cataldo, Wagstrom, Herbsleb, & Carley, 2006).

Dependencies in software development projects, and coordination mechanisms used to manage those dependencies, were the focus of Grinter’s (1996) work. She noted *“developers must manage a cadre of dependencies simultaneously if they are to build any working systems at all”* (Grinter, 1996, p. 50) and *“dependencies within the [software] code, create and reflect social dependencies that exist between developers, teams of programmers, and software development organisations”* (Grinter, 1996, p. viii). In her grounded theory study, she investigated software developers’ interactions with software configuration management systems. Table 4 summarises the dependencies found in the study. Dependencies were found at all organisational levels, but systems integration and external demands were the main source of dependencies in software development. For example, when a ‘build’ failed this caused considerable coordination work among the developers, testers, and technical writers. External demands included the influence of government regulations, vendor dependencies, and customer dependencies. Another finding was that tool support in the form of a system wide configuration management tools supports dependency management.

Table 4 Grinter's (1995) dependencies in software development

Level	Dependency	Description
Inter-organisational	Vendor	Reliance on vendor product functionality (e.g. operating system software) to support product functionality
	Customer	Requirements, support and enhancements
Group	Lifecycle	Multiple teams working in parallel on the same product for different platforms
	"Big picture"	Having an overview of how all parts of the system fit together
	Testing	Test suite versions must be managed
Individual	Parallel development	When two or more developers work in the same code module
	Change	Change in one part of a module requires changes in other modules, documentation, and test suites
	Expertise	Shared product knowledge
	Integration	Successfully assembling all the constituent modules of system
	Historical	Mining organisational memory for previous decisions or older code versions
	Configuration management tool	Developers come to depend on the tool for version control, problem tracking, task assignment, as a repository for data sources, but problems arise when the tool has not been updated accurately
	Interface dependencies	Many 'bugs' in the code base appear incorrectly as interface issues

Since software development has become globally distributed, further coordination issues have surfaced (Herbsleb, 2007). Inter-organisational coordination in software development is primarily concerned with remotely outsourced and geographically distributed projects. In distributed development coordination problems are exacerbated because many mechanisms for coordinating work in a co-located project are absent or inadequate (Herbsleb, 2007; Sabherwal & Choudhury, 2006). A large literature on issues of globally distributed development is available (Agerfalk & Fitzgerald, 2006; Crowston, et al., 2007; Domino, Hevner, & Collins, 2002; Espinosa, et al., 2007; Herbsleb, 2007; Herbsleb & Grinter, 1999).

Two perspectives on inter-organisational coordination in systems development are apparent, the technical perspective and the organisational perspective. Taking a technical perspective and focusing on research challenges, Herbsleb (2007) identifies 'coordination over distance' as a key phenomenon in global software development (GSD) and stated that "*global distribution of a project seriously impairs critical coordination mechanisms*" (Herbsleb, 2007, p. 1). He specifies the research challenges in this area as follows:

- Understanding the complex relationship between software dependencies and task dependencies in order to, for example, predict when co-location must be

used, understanding the most appropriate architectures for GSD, and understanding the importance of teams having a shared work history

- Measuring architectural/organisational fit and developing tactics to deal with adjusting the organisation or the architecture to improve this fit
- Learning how to support effective requirements negotiation
- Extending effective tools for coordination of co-located development to cope with GSD. Version control and change management tools are common in collocated development. For GSD, the tools set must be extended to include awareness of project members activities and expertise, additional communication abilities (e.g. chat, document sharing, shared editing sessions), and ways to exploit project memory
- Identifying “what practices are effective when?” (Herbsleb, 2007, p. 8), what practices are complementary and should be used together, and what practices are closely linked to specific problems and should be used whenever those problems occur. Such knowledge would allow knowledgeable tradeoffs between different practices

Dependencies in globally distributed software development were investigated by Espinosa, Slaughter, Kraut, & Herbsleb (2007). Based on Malone and Crowston's (1994) idea that coordination problems are caused by dependencies in a project, Espinosa, et al. (2007) investigated dependencies using a grounded theory approach based on interviews with software development team members (n=36) in a European telecommunications firm. They found technical, process, and temporal dependencies are the main types of dependencies, and they formulated a number of propositions linking coordination, software development, co-location and distributed development. They reported three findings. Coordination needs vary with the member's role. Geographic distance has a negative effect on coordination, mitigated by shared knowledge of the team and awareness of team member's presence. Shared task knowledge is more important for coordination among co-located members.

Taking an organisational perspective on inter-organisational coordination, Sabherwal and Choudhury (2006) explored governance of outsourced and globally distributed projects by investigating coordination and control mechanisms used in three case studies. They found that project performance is directly affected by coordination and control (see Figure 5).

Sabherwal and Choudhury (2006) defined control as a mechanism to ensure an individual's conformance to stated organisational or project goals, whereas coordination, as discussed in organisational theory, is a mechanism for managing the interdependencies between actors and activities that occur when the overall goal is broken down into activities and divided amongst actors.

Figure 5 Coordination and control (Sabherwal & Choudhury, 2006)

Sabherwal (2003) described the two streams of literature on these topics as distinct and provided a typology of coordination mechanisms based on a synthesis of coordination mechanisms identified in organisation theory, project management, and information systems (Adler, 1995; DeSanctis & Jackson, 1994; Galbraith, 1974; Kraut & Streeter, 1995; Thompson, 1967; Van de Ven, et al., 1976). His typology had four main coordination types: coordination by standards, coordination by plans, coordination by formal mutual adjustment, and coordination by informal mutual adjustment.

Although Sabherwal (2003) and Sabherwal and Choudhury (2006) distinguished between coordination and control, they also point out that what is considered a control mechanism by some (e.g. Kirsch (1997)), is considered a coordination mechanism by others (e.g. (Adler, 1995; DeSanctis & Jackson, 1994; Kraut & Streeter, 1995)).

Coordination and control both affect project performance (Sabherwal, 2003; Sabherwal

& Choudhury, 2006) but the relationship is not linear as *“improved coordination helps in the exercise of control while effective control may improve coordination”* (Sabherwal & Choudhury, 2006, p. 190).

Sabherwal (2003) found that coordination mechanisms, from the perspective of both clients and vendors, are influenced by six factors. Based on 11 cases of globally outsourced development they found the influences on coordination are complexity, criticality, uncertainty, efficiency, equity, and relational quality. Sabherwal and Choudhury (2006) built on this study, and showed that coordination mechanisms change over the life of a project.

A knowledge-based perspective of coordination in globally distributed development was taken by Kotlarsky, van Fenema and Willcocks (2008). Their perspective is that knowledge management is an important contributor to coordination, and that the more traditional concepts of information-processing (Galbraith, 1977; Thompson, 1967) are less appropriate because workers in the non-routine situation of product development are *“intelligent, learning, reflexive creative and, communicative knowledge workers”* (Kotlarsky, et al., 2008, p. 96). They questioned how coordination mechanisms facilitate knowledge processes, and used four main coordination mechanisms to structure their study: organisational design mechanisms, work-based mechanisms, technology-based mechanisms, and social (inter-personal) mechanisms. Organisational design includes formal coordination structures such as hierarchies, roles, and responsibilities. Work-based coordination includes the explicit knowledge embodied in plans, specifications, standards, and other artefacts of production. Technology-based mechanisms include the tools for managing information and communication such as scheduling, file version control, and asynchronous communication. Finally, social mechanisms are the cognitive and relationship-mediated practices used in knowledge work to share information.

To investigate their knowledge-based perspective, Kotlarsky, et al. (2008) carried out two case studies. The interviewed project teams in SAP in India and Germany, and Baan in India and the Netherlands to determine the types of knowledge and practices used to achieve coordination. They found, where many mechanisms are used to support knowledge processes the project was successful, and where there are very few the project was not successful. They assessed success as the adoption of the product in one case, and failure by the closure of the project in the other. They noted their research was designed with the assumption that one knowledge process is supported by a single communication mechanism, and that this may not be correct, *“it is possible that one*

coordination mechanism (or a combination) affects more than one knowledge process" (ibid, p. 107).

In summary, this section reviewing literature on coordination in IS project studies found:

- Coordination has been recognised as a critical element in effective software development since 1988 (Curtis, et al.)
- Quantitative IS research has explored coordination and project effectiveness, (reviewed more extensively in section 2.2.7 Project Success and Coordination)
- Software engineering research has focused on the relationship between system architecture, coordination, and dependencies (Grinter, 1996)
- Inter-organisational studies of software development show coordination and control, while they have different definitions, are frequently achieved using the same practices
- Coordination in studies of globally distributed development, has been categorised by coordination type (e.g. coordination by standards), by dependencies (e.g. technical, temporal, and process dependencies), or by organisation level coordination mechanisms (e.g. organisational design)

2.2.4 COORDINATION IN TEAMWORK STUDIES

Coordination studies of co-located teams and small groups recognises two forms of coordination, explicit and implicit (Nonaka, 1994; W. P. Wang, Kleinman, & Luh, 2001). Explicit coordination involves two or more team members sending communication messages to one another using formal or informal, oral or written, transactions to integrate their work. Implicit coordination occurs when team members anticipate the actions and needs of their colleagues and adjust their behaviour accordingly without preplanning or direct communication (Rico, Sanchez-Manzanares, Gil, & Gibson, 2008).

Explicit coordination is the focus in a study of teamwork quality in innovative software product development by Hoegl and Gemuenden (2001). Their teamwork quality construct included the following factors: communication, coordination, balance of member contributions, mutual support, effort, and cohesion. German software development teams (n=145) in two organisations provided the data in a standardised interview questionnaire of 575 team members, team leaders and managers. Teamwork quality directly and positively affected project efficiency and effectiveness, as well as team satisfaction and learning. Another quantitative investigation of coordination by

Yuan, Zhang, Chen and Vogel (2009) found explicit knowledge sharing and coordination technology had no significant impact, whereas implicit knowledge sharing significantly impacted coordination effectiveness in their study of 59 developer pairs.

Research on implicit coordination includes studies of shared mental models (Kang, Yang, & Rowley, 2006; Yuan, et al., 2009), collective mind (Crowston, Hala, Howison, & Chengetai, 2004; McChesney & Gallagher, 2004), and team expertise. Each is now considered.

A shared mental model (SMM) is a cognitive understanding common to all members of a team regarding the team member's skills, knowledge, and the tasks they perform. Such a model facilitates team communication and enhances performance, cooperation, cohesiveness, trust, group efficacy, satisfaction, commitment, and effectiveness (Kang, et al., 2006). *"In sum the SMM of team members promotes common expectations for team tasks and [team] member[s] and facilitates information processing and coordination"* (Kang, et al., 2006, p. 1688). Kang, Yang, and Rowley (2006) and Yang, Kang, and Mason (2008) confirmed this in their studies of the impact of SMM and other contingencies on team effectiveness, based on 277 surveys of 83 teams in 42 Korean software development companies. The contingencies they included were shared work history, age, tenure, and gender. Team effectiveness comprised measures of quality and quantity of team outputs and commitment. They found that a SMM significantly influences team effectiveness.

Collective mind is a concept developed by Weick and Roberts (1993) when investigating effective and ineffective practices on fighter carrier flight decks. They conceptualised collective mind as either heedful or heedless performance on the part of participants and determined that *"reliable performance may require a well developed collective mind in the form of a complex, attentive system tied together with trust"* (p. 378). Collective mind is used in studies of software development carried out on FLOSS (Free/Libre Open Source Software) teams (Crowston & Kammerer, 1998), and in outsourced development (McChesney & Gallagher, 2004) to describe coordination practices. In both cases collective mind theory was used along with Malone and Crowston's (1994) Coordination Theory. Collective mind was used because the authors considered it a better tool for exploring the implicit aspects of coordination.

Expertise coordination is a form of implicit coordination identified by Faraj and Sproull (2000) who divide coordination into administrative and expertise. Administrative coordination includes all of the formal explicit ways to assign tasks, allocate resources,

manage dependencies, and integrate outputs. Expertise coordination is the management of knowledge and skill dependencies, which becomes important when tasks are complex, non-routine, and intellectual. Such coordination is important for recognising where expertise is located, where it is needed, and where it can be accessed. The two coordination concepts were developed qualitatively using open-ended interviews of software developers, then verified quantitatively using a survey of 69 software development teams (333 respondents) in a single organisation to measure their impact on team performance (comprising team effectiveness and team efficiency). The presence of expertise, professional experience, administrative coordination, and the use of software methods had no effect on performance. Expertise coordination contributed significantly to both team effectiveness and efficiency, whereas administrative coordination contributed only to team efficiency. In addition, the authors noted that they omitted the constructs uncertainty and complexity from the study, which may moderate the relationship between expertise coordination and team performance.

Quantitative research linking team coordination and team effectiveness is extensive, covering general project teams and software development teams. The research does not provide a clear directional relationship between the two concepts. In some studies team effectiveness is argued to impact effective coordination (Nelson, 2007), but in most studies effective coordination is theorised to impact team effectiveness or team performance (Faraj & Sproull, 2000; Hoegl & Gemuenden, 2001; Nelson, 2007; Yang, et al., 2008). Coordination in these studies is operationalised in a number of different ways, as is team effectiveness. The section on Project Success and Coordination reviews project and team effectiveness measures.

This section has focused on studies of coordination in software development teams and shown that coordination can be explicit or implicit, and implicit coordination contributes to team effectiveness.

The following section completes the review of coordination by focusing on coordination studies undertaken in the context of agile development.

2.2.5 COORDINATION IN AGILE SOFTWARE DEVELOPMENT

A very small number of theoretical and empirical studies recognise coordination as a crucial aspect of agile software development. In earlier sections of this review on the literature on agile methods, some coordination concepts emerged in evidence from

ethnographic studies of agile projects. More recently, research has identified coordination's more central role.

Pries-Heje and Pries-Heje (2011) determined that coordination is one of four critical elements that explain why Scrum 'works' as a project management approach. Their single longitudinal case study of Scrum in a distributed project in Denmark and India found product backlog, sprint backlog, scrum board, and daily meetings were practices for achieving coordination.

A 'coordinator role' was identified by Hoda, Noble, and Marshall (2010) in their formal grounded theory study based on interviews of project team members in 14 software organisations using agile software development. A person taking this role "*acts as a representative of the self-organizing agile team to coordinate communication and change requests from customers*" (Hoda, et al., 2010, p. 288).

Research drawing on existing theories of coordination is also available. Cao and Ramesh (2007) used organisation theory to examine whether the coordination mechanisms suggested by agile methods (e.g. co-located customers, short iterations) were consistent with the coordination mechanisms proposed by Van de Ven et al. (1976) (i.e. coordination by impersonal, personal, and group modes). In their theoretical study, they proposed consistencies between the coordination modes from organisation theory and agile practices in small projects, but inconsistencies when using these agile methods in large projects because "*agile methods don't recognize the need for more impersonal coordination in large projects, which [organisational] coordination theory suggests*" (Cao & Ramesh, 2007, p. 44).

Thompson's (1967) organisation structure theory, involving interdependencies and coordination (i.e. coordination by standardisation, planning, and mutual adjustment), was used by Barlow, Giboney, Keith et al. (2011) to justify a theory-based methodology selection framework to guide large organisations in selecting between pure agile, pure plan-driven (Boehm & Turner, 2004), or hybrid methodologies. This study was theory-based, and was not tested empirically.

Coordination Theory (Malone & Crowston, 1994) was used by Pikkarainen et al. (2008) as a theoretical lens to study communication in two small co-located agile projects. This study 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.

Using a teamwork model with coordination as the outcome variable, Moe, Dingsoyr, and Dyba (2010) studied a single co-located Scrum project. In this project, Scrum practices were misapplied, partially because of the 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), which interfered with successful team coordination.

In summary, some theoretical and empirical research has made the link between coordination and agile software development. While these studies have described which practices act to achieve coordination, they have not attempted to develop theory with coordination as a central component.

2.2.6 A SUMMARY OF COORDINATION PERSPECTIVES

The preceding literature review identified various perspectives on coordination in the literature on organisation theory, Coordination Theory, IS projects, teamwork, and agile software development. This section summarises those perspectives according to the following categories: structural, architectural, technological, spatial, and cognitive coordination, coordination by artefacts, and coordination by activities. Each is summarised as follows.

Structural coordination is organisational level coordination reflecting the structure or design of an organisation (Hatch & Cunliffe, 2006; Mintzberg, 1980). Organisational structure includes formal coordination structures such as hierarchies, roles, and responsibilities (Kotlarsky, et al., 2008). Forms of coordination identified in the organisation studies literature included coordination by impersonal mode (by rules and procedures), and by mutual adjustment (group – by meetings, or personal – horizontal or vertical) (Van de Ven, et al., 1976). Agile development approaches seem to adhere most closely to the mutual adjustment type of structural coordination.

Architectural coordination is coordination imposed by the system architecture on the interactions of the development team (Herbsleb & Grinter, 1999). Architecture is considered an important coordination mechanism in software engineering (Herbsleb, 2007), but agile methods provide no mechanisms for addressing system architecture issues (Nord & Tomayko, 2006).

Technological coordination is coordination achieved using software tools. This includes specific tools such as integrated software development environments (Barthelmeß, 2003; Grinter, 1996), and generic tools such as email, social networking applications,

and project management software. Within the agile methods, only XP states that tools are a fundamental requirement for successful method adoption (i.e. automated environments for test-driven development and regression testing) (Beck, 2000). No research was found linking tool use with agile project coordination.

Spatial coordination is coordination achieved through the arrangement of artefacts or actors. This form of coordination includes practices such as co-location of teams, or seating developers at a single workstation (e.g. during pair programming), common practices in agile development (MacKenzie & Monk, 2004; Sharp & Robinson, 2008).

Cognitive coordination occurs within work groups without explicit speech or message passing. Cognitive coordination is also called implicit coordination, which occurs when team members anticipate the actions, needs, or understandings of their colleagues and adjust their behaviour accordingly without preplanning or direct communication (Rico, et al., 2008). Shared mental models (Kang, et al., 2006), collective mind (Crowston, et al., 2004; McChesney & Gallagher, 2004; Weick & Roberts, 1993), and expertise coordination (Faraj & Sproull, 2000) are all forms of implicit coordination. Few examples from the agile development literature could be located that focus on cognitive coordination. Although one conference article focused on 'mindful' relationships in agile teams (Matook & Kautz, 2008). Given agile development's emphasis on teamwork and close collaboration, it seems likely that cognitive coordination would contribute in some way to coordination within agile software development projects.

Coordination by artefacts is coordination mediated by artefacts in a work process. For example, artefacts in the form of publicly visible whiteboards are used for coordinating staff activities within hospitals (Ren, Kiesler, & Fussell, 2008). In artificial intelligence, coordination can be achieved using 'blackboard architecture', which is when program modules interact by searching a global blackboard for their inputs and posting their outputs on the same blackboard (Malone & Crowston, 1994). Wallboards and story cards are typical coordination artefacts in XP (Sharp & Robinson, 2008).

Coordination by activities is coordination achieved by individual or group actions. The ethnographic studies of agile software development identified a number of coordinating activities including, in XP projects - frequent meetings, pair programming, and refactoring (Robinson & Sharp, 2004; Sharp & Robinson, 2004; Sharp & Robinson, 2008), and in Scrum projects - daily stand-up meetings (Moe, et al., 2008).

Two further potentially relevant perspectives on coordination were identified in the literature: temporal and relational coordination. They are presented here, because they did not readily fit into earlier sections of the review.

Temporal coordination is an activity that ensures collaborative work can be achieved at the appropriate time (Bardram, 2000). Based on a study in a Danish hospital surgical department, Bardram (2000) found that temporal coordination is mediated by temporal coordination artefacts such as operation schedules. He also proposed three subtypes of temporal coordination, synchronisation (time ordering of events), scheduling (planning of the time ordering of events), and allocation (allocating duration to events). In agile software development, temporal coordination is most apparent in the iterative (repetitive) nature of the process. Each iteration, is a miniature development lifecycle with a specific duration, or time-box. Within each time-boxed iteration, team members carry out concurrent and sequential activities (Beck, 2000; Schwaber & Beedle, 2002; Stapleton, 1997).

Relational coordination is coordination mediated by human interrelationships. It is about the coordination between people's assigned roles, rather than between individuals or tasks. (Gittell, Seidner, & Wimbush, 2010). Relational coordination theory was developed in the context of hospitals and airlines (Gittell, 2011), and is defined as "frequent, timely, problem-solving communication" (Ren, et al., 2008, p. 108). Relational coordination would appear to be relevant in agile software development, which achieves relational coordination by having project teams working together alongside business people on a daily basis, face-to-face communication, and regular team meetings for problem solving (Beck et al., 2001).

The key findings from this review of the literature on coordination can be summarised as follows. Coordination concepts in organisation studies take an organisation-level view that may not be relevant at the project level, where agile software development occurs. Coordination Theory deals primarily with explicit coordination, and is not suitable for prediction, although it is useful for identifying dependencies and coordination mechanisms. IS project literature draws on coordination concepts from organisation studies and Coordination Theory. That literature highlights the critical importance of coordination in IS projects, the influence of system architecture on coordination in IS projects, and the coordination problems encountered in globally distributed software development. Teamwork studies provide evidence for implicit coordination, but only a single study was found proposing that implicit coordination

occurs in an agile software development context. Furthermore, there are various perspectives on coordination in the extant literature, but none of them have been applied to explain or predict the effectiveness of agile software development.

Systems development methodologies (and agile software development is a distinct class of SDM) are adopted because of the assumption that they contribute to project success. Well-coordinated projects are also assumed to contribute to project success. Therefore, the next section focuses on the predictive literature on IT and IS project success, and the place of coordination in project success.

2.2.7 PROJECT SUCCESS AND COORDINATION

Research on project success occurs in three areas pertinent to a study of agile software development: information systems (IS), IS project management, and IS development. Each of these areas addresses project success differently.

Success in Information Systems

In information systems, success is commonly evaluated using one or more components of the 'IS success model' developed by DeLone and McLean. Their first model was published in 1992, elaborated in 2003, and again in 2008 (DeLone & McLean, 1992; DeLone & McLean, 2003; Petter, DeLone, & McLean, 2008). The 2008 model contains six concepts: system quality, information quality, service quality, use, user satisfaction, and net benefits. Relationships between each of the constructs have been independently tested and shown to be valid (Petter, et al., 2008).

The DeLone and McLean models (DeLone & McLean, 1992; DeLone & McLean, 2003; Petter, et al., 2008) focus on the success of the IS product (an information system), but as Aladwani says, "*it is important to understand that IS project performance is a different construct from IS effectiveness*" (Aladwani, 2002, p. 187). He considered four ways in which the two constructs differ. IS project performance focuses on the project, whereas IS effectiveness focuses on the information system. IS projects have their own unique social context, which differs from that of the IS system. IS projects have the project team as the unit of analysis, whereas IS systems tend to have the individual as a unit of analysis. Finally, IS project effectiveness is measured with multi-dimensional constructs such as project efficiency or effectiveness, whereas IS systems tend to be measured with one-dimensional measures such as IS use or IS quality. Based on this argument, Aladwani (2002) formulated and tested a project effectiveness model based on concepts in project management, information systems, and organisational team

literature (summarised in Table 5). His study concluded by saying, “*researchers are encouraged to improve upon our theory by incorporating other constructs, such as, participation, coordination techniques, the newness of the technology, and the familiarity of the staff with the technology implemented, to name a few*” (Aladwani, 2002, p. 204). In this statement, Aladwani recognises the potential contribution of coordination to project effectiveness (or success). Furthermore, his arguments show that IS success is not the same as IS project success. Therefore, measures of IS success are not appropriate for studying IS projects. Based on this argument, IS success measures are inappropriate for studying agile software development projects, which are a form of IS project.

Success in IS Project Management

Studies in IS project management research tend to measure project success with a small number of outcome variables. These outcomes include scope, quality, cost, time, and performance. Project management authors often refer to these outcomes as an ‘iron triangle’. What constitutes the triangle varies. For example, Kerzner (2003) uses time, cost, and performance, when discussing general project management, Schwalbe (2010) uses time, cost, and scope when discussing IT project management, and Atkinson (1999) uses time, cost, and quality as the measures of IS project success. Using these variables as measures of project success is criticised because of their narrow metrics-based focus and because they do not take into consideration stakeholder satisfaction (Jugdev & Muller, 2005). For example, a project may fail to meet any one of these criteria, or possibly all of them, and yet aspects of the project may still be considered successful. Alternatively, a project may meet the criteria of time, cost, and scope, and yet its customers may not be satisfied with the final system. Other success criteria are argued to be more useful (Agarwal & Rathod, 2006; Dvir, Lipovetsky, Shenhar, & Tishler, 1998; Jugdev & Muller, 2005). Jugdev and Muller (2005) found stakeholder views of success are important in assessing project success in their review of general project management success factors from the 1960s to 2004. They found that acceptance and usability are the key success measures for project deliverables. In concluding their review they said, “*project success is a complex and ambiguous concept and it changes over the project and product life cycle*” (Jugdev & Muller, 2005, p. 29). A study of IT project managers notions of project success resulted in similar conclusions. Thomas and Fernandez surveyed 36 Australian companies to determine how they define IT project success and found “*there was no one best method for defining and measuring success*” (Thomas & Fernandez, 2008, p. 739).

Success and Coordination in IS Development Research

Success studies in IS development research find a relationship between project coordination and success. Table 5 summarises the major IS studies addressing these topics. The studies form two groups, those investigating antecedents of project success, and those investigating antecedents of team success. Project-based studies tend to measure project performance in terms of software product and or process success (Barki, et al., 2001; Kraut & Streeter, 1995; MacCormack, et al., 2001; Nidumolu, 1995). Team-based studies measure team effectiveness in a variety of ways, and have incorporated a variety of different antecedent variables (Faraj & Sambamurthy, 2006; Hoegl & Gemuenden, 2001; Kang, et al., 2006). What these studies show is the multi-dimensional nature of the project success construct, and the lack of consensus on what constitutes project success.

What the studies in Table 5 do show is the generally positive influence of coordination on IS project success (in particular, Andres & Zmud, 2001; Chen, et al., 2009; Hsu, Shih, Chiang, & Liu, 2012; Nidumolu, 1995; Parolia, Goodman, Li, & Jiang, 2007).

Coordination in these studies is typically defined using the information-processing view of organisation theory (i.e. coordination is horizontal or vertical). Project success is measured in various ways including as project performance, project team performance, residual performance risk, process performance, and product performance .

Table 5 Empirical studies of project success

Dependent variables UoA - Unit of analysis Source	Antecedent variables (a) Moderating variable (m) Mediating variable (me) Control variable (c)	Aim Issues Key results
<p>Coordination success Managers' evaluation – process and product Client satisfaction Software productivity Software quality Project members informed</p> <p>UoA = Software development project</p> <p>Source (Kraut & Streeter, 1995)</p>	<p>Project age (a) Project size (a) Organizational interdependence (a) Project certainty (a) Impersonal procedures (a) Formal interpersonal procedures (a) Informal interpersonal procedures (a) Electronic communication (a) Interpersonal network (a)</p>	<p>Aim - investigates success of various coordination techniques Issue - Projects using waterfall development process Results - Project coordination correlates strongly with customer satisfaction Project size positively predicts use of formal coordination Interpersonal networks are more beneficial when projects are uncertain Large projects also need formal coordination</p>
<p>Process performance (process flexibility, process predictability Competitive performance</p> <p>UoA = IT project</p> <p>Source (Nidumolu & Subramani, 2004)</p>	<p>Standardization of methods (a) Decentralisation of methods (a) Standardization of performance criteria (a) Decentralization of performance criteria (a) Firm size (a)</p>	<p>Aim – investigates control strategies used by software development firms Issue – sample of 56, matched sample from marketing and software development Results - Decentralization of methods is positively related to process performance Standardization of performance criteria is positively related to process performance Software development process performance is positively related to competitive performance</p>
<p>Product quality (as a measure of project performance)</p> <p>UoA = software project</p> <p>Source</p>	<p>Investment in architectural design (a) Early market feedback (a) Early technical feedback (a) Generational experience (a) Resources (c)</p>	<p>Aim - investigates the characteristics of an effective development process in uncertain and dynamic environments Issue - using flexible iterative software development process. Expert panel to assess project performance Results - Allocating resources to design of product architecture results in higher quality products</p>

Dependent variables UoA - Unit of analysis Source	Antecedent variables (a) Moderating variable (m) Mediating variable (me) Control variable (c)	Aim Issues Key results
(MacCormack, et al., 2001)		Early market feedback improves product quality Early technical feedback results in higher product quality Generational experience and resources can be traded off
Project performance UoA = IS project Source (Chen, et al., 2009)	Pre-project partnering (a) Vertical coordination (a) Horizontal coordination (a) Perception gap (a)	Aim – examines relationship between pre-project partnering, horizontal and vertical coordination and user—developer requirements perception gaps Issue – respondents comprised project managers belonging to PMI in USA only. No single project reported. Results - Pre-project partnering has a positive effect on vertical and horizontal coordination Horizontal and vertical coordination have a positive effect on project performance User –developer perception gap mediates project performance
IS project performance (task outcomes, psychological outcomes, organisational outcomes) UoA = Project team Source (Aladwani, 2002)	Process characteristics (problem solving competency) (a) Technology characteristics (support) (m) Task characteristics (goals) (m) Organisational characteristics (advocacy) (m) Project characteristics (team size) (m) People characteristics (expertise) (m)	Aim – builds an integrated model of IS project performance including major project contingencies, both technical and social Issue – concepts based on literature in project management, organisation teams, and information systems Results - Problem solving competency effects project performance Management support effects task and organisational outcomes Staff expertise effects process and organisational outcomes Support technologies effect task outcomes Clarity of project goals effects process and psychological outcomes Project team size negatively impacts problem solving
Project performance Residual performance risk Process performance Product performance UoA = IT project Source	Project uncertainty (requirements and technical) (a) Vertical coordination (a) Horizontal coordination (a)	Aim – studies the effects of coordination mechanisms and project uncertainty on software project performance Issue – data primarily from banking industry. Data gathered 17 years ago from completed projects (retrospective recall) Results - Horizontal coordination has positive effect on project performance Vertical coordination reduces project uncertainty and residual project risk Higher levels of both vertical and horizontal coordination led to higher

Dependent variables UoA - Unit of analysis Source	Antecedent variables (a) Moderating variable (m) Mediating variable (me) Control variable (c)	Aim Issues Key results
(Nidumolu, 1995)		levels of project performance Project uncertainty reduces performance
Project performance (process performance (cost gap) & product performance (quality)) UoA = Software project Source (Barki, et al., 2001)	Fit (a) Risk exposure (a) Risk management (formal planning, internal integration, user participation) (a)	Aim – determines if software project performance is influenced by the fit between a project's risk management profile and its risk exposure Issue – authors state that IS researchers must not consider projects to be the same as organisations and that organisation theory concepts and hypotheses may not be accurate in assessing project studies Results - The farther a project's risk management deviates from an ideal profile (based on internal integration, user participation and formal planning) the lower its performance in terms of budget compliance and system quality Higher levels of uncertainty call for lower levels of formal planning (contrary to information-processing theory)
Project performance (software quality, meeting targets) UoA = Software project Source (Deephouse, Mukhopadhyay, Goldenson, & Kellner, Winter 1995-96)	Software process (project planning, process stability, process training, coordination with customer, design reviews, prototyping, cross-functional teams) (a) Project characteristics (size, stage, domain) (m) Rework (m)	Aim - investigates effectiveness of software processes Issue – Respondents from software engineering process improvement group (possible bias) Results - Process planning, effects meeting schedule, budget and product quality Process training improves planning effectiveness Cross-functional teams effect product quality and planning effectiveness Inconclusive results for effect of design reviews, prototyping, frequent user contact and maintaining a stable environment
Software development success (team productivity, process satisfaction) UoA = IS student Source (Andres & Zmud, 2001)	Task interdependence (a) Coordination strategy (mechanistic, organic) (a) Goal conflict (a)	Aim – Investigates the affect of task interdependence, goal conflict, and coordination strategies on productivity and satisfaction associated with software design and coding activities Issue – experiment, factorial 2*2*2 with senior IT students, students paid money to participate, function points used to equalise tasks assigned Results - Organic coordination strategy leads to greater project success (team productivity) than mechanistic strategy When task interdependence is high the effect or organic coordination on project success increases

Dependent variables UoA - Unit of analysis Source	Antecedent variables (a) Moderating variable (m) Mediating variable (me) Control variable (c)	Aim Issues Key results
		An organic coordination strategy is associated with greater software development process satisfaction Goal conflict was not a significant factor in the study
Software performance risk Process control Product flexibility UoA = Data processing unit Source (Nidumolu, 1996)	Requirements uncertainty (a) Vertical coordination (a) Horizontal coordination (a)	Aim – studies the effect of project coordination and requirements uncertainty on process control and product flexibility Issue – uses same data set as Nidumolu (1995) Results - Vertical coordination reduces performance risks Horizontal coordination leads to flexible software applications
Project performance UoA = IS team Source (Parolia, et al., 2007)	Horizontal coordination (a) Vertical coordination (a) Leadership commitment and empowerment (me) Knowledge transfer (me) Clear mission and objectives (me)	Aim – studies how horizontal and vertical coordination affect the performance of IS projects Issue – none Results - Horizontal coordination has positive effect on mission and objectives Vertical coordination has positive effect on mission and objectives Vertical coordination leads to knowledge transfer in teams Horizontal coordination leads to knowledge transfer in teams
Project team performance UoA = IS teams in Taiwan Source (Hsu, et al., 2012)	Coordination (a, me) Communication (a, me) Transactive memory system (TMS) (a)	Aim – studies how coordination, communication, and transactive memory systems affect project team performance in IS projects Issue – none Results - TMS has a positive effect on project team performance TMS has a positive effect on coordination TMS has a positive effect on communication Coordination has a positive effect on project team performance Communication has a positive effect on project team performance
Team effectiveness Team commitment Team performance UoA = Software team	Shared mental model (a) Age (a) Tenure (a) Gender (a) Task (a)	Aim – studies the effects of team member cognitive and demographic characteristics on team effectiveness Issue – none Results - Cognitive similarities are more important than demographic similarities in

Dependent variables UoA - Unit of analysis Source	Antecedent variables (a) Moderating variable (m) Mediating variable (me) Control variable (c)	Aim Issues Key results
Source (Kang, et al., 2006)	Member (a) Team size (c) History (c)	team effectiveness
Team performance UoA = Software team Source (Faraj & Sambamurthy, 2006)	Directive leadership (a) Empowering leadership (a) Team experience (m) Task uncertainty (m) Administrative coordination (c) Team size (c)	Aim – studies the effects of team leadership and coordination on team performance Issue – stakeholders perceptions used Results - Leadership is independent of coordination mechanisms Empowering leadership positively impacts team performance under conditions of high team expertise and high task uncertainty
Team performance Effectiveness Efficiency Personal success Work satisfaction Learning UoA = Software development teams Source (Hoegl & Gemuenden, 2001)	Teamwork quality (a) (Communication, Coordination, Balance of member contributions, Mutual support, Effort, Cohesion) (a)	Aim – defines teamwork quality by investigating the relationship of teamwork in software teams to project success Issue – team member, team leader, manager perceptions used, results differ by role Results - Quality of interactions is more important than activities Team work quality is significantly related to project success (team performance & personal success)
Team potency Development cost Speed to market Market success UoA = Software team Source (Akgun, Keskin, Byrne, & Imamoglu, 2007)	Goal clarity (a) Team experience (a) Intra-team trust (a) Team empowerment (a) Team size (a)	Aim – investigates the impact of ‘team potency’ on software development project success Issue – None Results - Intra team trust, team experience and team empowerment have a positive association with team potency Team potency is positively associated with speed to market, low development cost and market success

A single study of software development projects was found that focused on coordination success as a criterion variable. That study measured the impact of horizontal and vertical coordination techniques, and project characteristics (project uncertainty, organizational interdependence, and project age, size, and stage) on coordination success (Kraut & Streeter, 1995). Coordination success was assessed as the degree to which a project was perceived as well coordinated. Results were consistent with the literature on coordination in organisations, in particular Van de Ven et al.'s (1976) findings. The results showed that informal interpersonal communication (i.e. horizontal coordination) was necessary for coordination under conditions of uncertainty, and when a project was more inter-organisationally interdependent. Vertical coordination did not predict project coordination success.

This section highlights three points. Firstly, IS effectiveness focuses on the success of the IS product rather than the successful development of that product, and therefore IS success measures are not appropriate measures of IS *project* success. Secondly, research in project management and IS development indicates project success is assessed using a variety of different measures. Finally, evidence from multiple studies of software projects shows a positive relationship between project coordination and project success. A single study of software development projects was found that investigated the impact of coordination techniques on coordination success.

This suggests studies of coordination can focus on *project coordination effectiveness* as the criterion variable, rather than project success, because project coordination has been shown to contribute to project success. This argument is incorporated into the conceptual framework discussed in the next major section of this review.

Factors Influencing IS Project Coordination

The articles summarised in Table 5 show that many factors impact project coordination and project success. The literature on agile software development highlights five factors as being especially important to the success of agile development projects, although their exact effects have not been measured. They include team size, project complexity, project uncertainty, task interdependence, and expertise (Cao & Ramesh, 2007; Cockburn, 2002; Little, 2005; Ramesh, Abrahamsson, Cockburn, Lyytinen, & Williams, 2005). Each of these factors is now considered.

Team size is a critical variable in all agile methods. The normative literature argues for a maximum team size of 10 people (Beck, 2000; Cockburn, 2002; Highsmith, 2002; Schwaber & Beedle, 2002; Stapleton, 1997). Recent research supports this conjecture

showing that projects with teams of nine or more, are less productive than smaller teams (Rodriguez, Sicilia, E, & Harrison, 2012). In agile software development when teams must be larger than this optimum, special adjustments are made, such as implementing a 'scrum of scrums' (a project divided among multiple teams) (Schwaber & Beedle, 2002), or adjusting the 'methodology weight' (amount of formality and documentation) (Cockburn, 2002). Team size has been used to indicate when to move from agile to more formal, plan-driven methods. That is, when team size is small, agile methods are considered most appropriate, but as team size increases a plan-driven method is more appropriate (Boehm & Turner, 2004). In the quantitative studies in Table 5, the literature is equivocal. For example, in Kraut and Streeter's (1995) study, project size influences project coordination, because larger projects are associated with increased use of vertical (more formal) coordination. Other studies found team size has no effect on project performance (Deephouse, et al., Winter 1995-96). One study found team size reduced problem solving ability (Aladwani, 2002), but others found team size had no reported effect on team effectiveness (Akgun, et al., 2007; Faraj & Sambamurthy, 2006; Kang, et al., 2006).

Complexity is 'problem size' according to the author of Crystal methods, one of the agile methods. Cockburn (2002), defines problem size as "*the number of elements in the problem and their interdependencies*" (Strode, 2005, Appendix K, p. 51). He uses this factor to determine which form of agile method to adopt.

Complexity is an amalgamation of a number of concepts. Xia and Lee (2005) identified 15 distinct items contributing to complexity in the IS literature, after carrying out an empirical study involving 541 project managers. All of the items are shown in Table 6. An analysis of Xia and Lee's (2005) results, shows that complexity is comprised of items related to change and interdependency. That is, their complexity construct includes eight items involving change (e.g. the end-users' business processes changed rapidly), and seven items involving interdependence (e.g. the project involved multiple technology platforms). This analysis is shown in the column labelled 'category' in the table. The outcome of this analysis is used in the conceptual framework discussed in the next section.

Uncertainty is linked with change in the IS literature. Change is an important consideration in the agile methods, particularly XP. The seminal book by Beck (2000) describing this method is entitled *Extreme Programming Explained: Embrace Change*,

which indicates the chief purpose of the method is to accommodate constant change while maintaining project progress.

Table 6 Complexity items in ISD projects adapted from Xia and Lee (2005)

Information systems development project component	Item description	Category
Structural organisational complexity	1. The project team was cross-functional	I
	2. The project involved multiple external contractors	I
	3. The project involved coordination of multiple user units	I
Structural Information Technology complexity	4. The system involved real-time data processing	I
	5. The project involved multiple software environments	I
	6. The project involved multiple technology platforms	I
	7. The project involved a lot of integration with other systems	I
Dynamic organisational complexity	8. The end-users' organizational structure changed rapidly	C
	9. The end-users' business processes changed rapidly	C
	10. Implementing the project caused changes in the users' business processes	C
	11. Implementing the project caused changes in the users' organizational structure	C
	12. The end-users' information needs changed rapidly	C
Dynamic Information Technology Complexity	13. IT architecture that the project depended on changed rapidly	C
	14. IT infrastructure that the project depended on changed rapidly	C
	15. Software development tools that the project depended on changed rapidly	C
Key I – item is a form of interdependency C – Item is a form of change		

Madsen notes, “*change and uncertainty are mentioned as fundamental characteristics of ISD⁷ in almost every contribution within the field*” (Madsen, 2007, p. 856). Barki et al. (1993) consider uncertainty factors to be equivalent to risk factors and argue that “*risk and uncertainty factors, as discussed in the IS literature, are one and the same, and should all be named uncertainty factors*” (1993, p. 207).

Uncertainty is an amalgamation of a number of concepts. Barki et al. (1993) identified 35 variables related to uncertainty in the IS literature. They operationalised their measurement using 144 items and tested them for validity across 120 software

⁷ ISD – Information systems development

development projects in 75 organisations. Their results showed 23 statistically significant uncertainty variables, as shown in Table 7. They grouped these variables into five factors: technological newness, application size, lack of expertise, application complexity, and organisational environment.

An analysis of Barki et al.'s factors, shows that uncertainty includes variables associated with complexity (e.g. task and technical complexity), size (e.g. team size), interdependency (e.g. number of links to existing systems), and expertise. This analysis is shown in the column labelled 'category' in Table 7. The outcome of this analysis is used in the conceptual framework discussed in the next section.

Table 7 Uncertainty factors adapted from Barki, Rivard and Talbot (1993)

Software development uncertainty factors	Variables	Category
Technological newness	1. Need for new software	O
	2. Number of software suppliers	I
	3. Need for new hardware	O
	4. Number of hardware suppliers	I
	5. Number of users outside the organisation	I
Application size	6. Team diversity	O
	7. Number of people on team	S
	8. Number of users in organisation	S
	9. Relative project size	S
	10. Number of hierarchical levels occupied by users	O
Lack of expertise	11. Team's lack of general expertise	E
	12. Lack of development expertise in team	E
	13. Team's lack of expertise with task	E
	14. Team's lack of expertise with application	E
	15. Lack of user experience and support	E
Application complexity	16. Number of links to future systems	I
	17. Number of links to existing systems	I
	18. Technical complexity	Com
Organisational environment	19. Extent of changes	C
	20. Intensity of conflicts	O
	21. Lack of clarity of roles definitions	O
	22. Resource insufficiency	O
	23. Task complexity	Com
Key I – variable is a form of interdependency C – variable is a form of change Com – variable is a form of complexity S – size variable E – expertise variable O – other variable		

Although the concepts of change and uncertainty are tightly coupled in the IS development literature (Lyytinen & Newman, 2008; Madsen, 2007), agile methods literature focuses on coping with change (Beck, et al., 2001a) rather than uncertainty. This thesis takes the IS viewpoint, and considers the two as closely related, since change might lead to increased uncertainty (e.g. a change in a requirement might lead to uncertainty as to how to best incorporate this new requirement into the existing software). Therefore, uncertainty is likely to be an important factor in agile software development projects, just as it is for other types of software project.

Task interdependence is important in organisational structure studies (Van de Ven, et al., 1976), as discussed in section 2.2.1 Coordination in Organisation Studies. The concept appears in many studies of coordination, because task interdependence increases the need for coordination in work units (Albino, Pontrandolfo, & Scozzi, 2002; Andres & Zmud, 2001; Kraut & Streeter, 1995; Levitt, et al., 1999; Malone & Crowston, 1994; Rico, et al., 2008). There have been few references made to interdependence in agile methods publications. Although Crystal methods use the number of dependencies to determine 'problem size', a factor in determining which Crystal method is appropriate in a given situation (Cockburn, 2002). Furthermore, Cao and Ramesh (2007) conjectured that task interdependency is high in agile software development projects when they proposed that organisational coordination theory can partially explain why agile software development practices are effective.

Expertise is another important factor in IS development studies, teamwork studies, and agile methods. Expertise is a multidimensional construct and includes various types of experience. Expertise includes generational experience (how many software projects a person had worked on previously (Akgun, et al., 2007; Faraj & Sambamurthy, 2006; Kang, et al., 2006; MacCormack, et al., 2001), duration (how long the team have worked together (Yang, et al., 2008)), technical and other relevant skills (Faraj & Sproull, 2000), and domain knowledge (Deephouse, et al., Winter 1995-96).

In agile method normative literature (listed in Table 1), expertise is considered an important, but scarce, quality. To reduce the need to rely on high levels of individual expertise, knowledge is shared among a team by using specific agile practices (e.g. co-location of teams, pair programming, and daily meetings) (Beck, 2000; Beck et al., 2001b; Cockburn, 2002). Team duration is also important because it improves individual expertise by supporting the transfer of technical and project knowledge among team members (Nerur & Balijepally, 2007; Schwaber & Beedle, 2002).

This section has shown that complexity, change, size, interdependence, and uncertainty are important in agile software development literature. They are also important in IS project literature. IS project research has defined them in terms of each other, as was highlighted in Table 6 and Table 7. An analysis showed that Xia and Lee (2005) defined complexity as a combination of change and interdependency, whereas Barki et al. (1993) defined uncertainty as a combination of change, interdependency, complexity, expertise, and size. This information is used in the conceptual framework discussed in the next section.

In summary, this review of project success and coordination literature highlights the following.

- IS success is not the same as IS project success; the focus is different, and therefore IS success measures are not appropriate for studying agile software project success.
- ISD research provides a more nuanced picture of project success, and the part coordination plays in success, than project management literature does.
- IS project success is a multi-dimensional construct, which is operationalised in different ways in different studies.
- Coordination (based on formality/informality and horizontal/vertical dimensions) is associated with IS project success in a number of quantitative studies of software teams and projects.
- There is evidence that IS project coordination contributes to IS project success. Therefore, studies of coordination can use *project coordination effectiveness* as the criterion variable.
- Very few measures of IS project coordination effectiveness are available, and none are developed to measure effectiveness in agile software development projects.
- Many contingencies impact IS project effectiveness. Those associated with the agile development approach are team size, complexity, uncertainty, expertise, and interdependence. IS literature defines and operationalises these contingencies, however examination shows their definitions are not mutually exclusive:
 - Complexity includes change and interdependency
 - Uncertainty includes change, interdependency, complexity, expertise, and team size

The final sections of this chapter describe the conceptual framework for this study developed from this review of the literature. Then the research questions are justified and presented.

2.3 CONCEPTUAL FRAMEWORK

A conceptual framework broadly defines concepts, constructs, or variables of interest, proposed relationships among them, boundary constraints, and delimitations. Consequently such a framework defines what will be studied (Gregor, 2006). The conceptual framework for this study draws on the literature reviewed previously, including research on the agile development approach, organisational coordination theories, Malone and Crowston's (1994) Coordination Theory, IS project coordination, teamwork, and project success.

Another contributor to the conceptual framework of this study is a theoretical framework explaining teamwork coordination proposed by Espinosa, Lerch, and Kraut (2004). Espinosa et al.'s (2004) framework is introduced in this section rather than the literature review, because these authors meld the coordination research presented in different sections of the literature review into a single framework. Their framework is shown in Figure 6**Error! Reference source not found.** In particular, Espinosa et al. (2004) draw on Malone and Crowston's (1994) Coordination Theory, and coordination concepts in organisation studies and teamwork literature. Espinosa et al.'s (2004) framework proposes that various task dependencies are managed with coordination mechanisms. Espinosa et al. (2004) propose that these mechanisms can be implicit, such as shared mental models, or explicit, such as plans and schedules. Coordination mechanisms interact together in a process of 'coordinating' to bring about a 'state of coordination'. The key idea is that *"an effective strategy for coordination success (i.e., high state of coordination) involves finding a mix of coordination mechanisms well-suited for the task"* (Espinosa, Lerch, & Kraut, 2002, p. 32). Espinosa et al.'s (2004) framework further argues that a state of coordination leads to a level of team performance and, finally, a state of coordination is one of many antecedent variables contributing to team performance.

In its current form Espinosa et al.'s (2004) framework is not completely suitable for a study of co-located agile software development projects because it is based on, and designed for, studies of teams in asynchronous (i.e., non-real time) and geographically dispersed contexts. The study in this thesis focuses on projects in synchronous and co-

located contexts because this is the environment that is most appropriate for agile software development. Co-located and synchronous environments may involve different dependencies and different coordination mechanisms for managing them than the distributed and asynchronous environments upon which Espinosa et al.'s (2004) framework is based.

Figure 6 Espinosa et al.'s (2004) framework

Nevertheless, the basic assumptions proposed in Espinosa et al.'s (2004) framework apply to the current study also, namely that explicit and implicit coordination mechanisms interact in a process of 'coordinating' to bring about a 'state of coordination'. In this study, these two concepts are named respectively 'coordination strategy', and 'coordination effectiveness'. This is to make a clearer distinction between their different functions. A coordination strategy is what you do to achieve coordination, whereas coordination effectiveness is the extent to which that coordination strategy addresses project dependencies.

Malone and Crowston's (1994) Coordination Theory is useful for the purposes of this study, because it provides a broader conceptualisation of coordination than that of traditional IS project management. Coordination Theory enables the investigation of many different coordination aspects of a project. It is not based around the role of a single person, the project manager. In traditional project management, the project manager's role is to coordinate a project (Kerzner, 2003; Napier, Keil, & Tan, 2009), but

some aspects of coordination may be outside the realm of a project manager's interest or influence (e.g. code module dependencies). Furthermore, agile software development may be incompatible with traditional project management ideas and practices (Boehm & Turner, 2004).

Coordination Theory is also useful because it allows for a detailed study of coordination in any context, and at any level: individual, group, or organisational. This is not the case with coordination in organisation studies. Organisational coordination concepts broadly describe coordination within business functional units (as discussed in section 2.2.1 Coordination in Organisation Studies). This study focuses on project-based work practices at the individual and group level, and organisational coordination concepts may be less suitable in this context.

Furthermore, Coordination Theory is based on the idea that coordination mechanisms are used to manage, or address, dependencies in a situation. This idea is central to the conceptual framework of this study and provides the basis for two fundamental concepts uniquely defined for the purposes of guiding this study as follows:

1. A dependency occurs when the progress of one action relies upon the timely output of a previous action, or the presence of some specific thing.
Dependencies lead to potential or actual constraints on projects.
2. A coordination mechanism is an entity (person or artifact) or activity (practice or technique) addressing one or more dependencies in a situation.

In agile software development projects, as in any projects, a number of different dependencies may exist, and a variety of different coordination mechanisms may be used to address those dependencies. This study uses this idea, and proposes that coordination mechanisms in agile software development projects consist of various agile method practices (e.g. pair programming, sprint backlog, information board) along with other development practices (e.g. configuration management tools) not specific to agile methods. All such coordination mechanisms may contribute to project coordination, not just those that are agile software development practices. This leads to another definition:

3. A coordination strategy is a group of coordination mechanisms used in a situation (or project, in the case of agile software development). Coordination mechanisms form a strategy because they are selected consciously by project stakeholders, rather than occurring by chance.

Based on these arguments, the conceptual framework guiding this study distinguishes two concepts: coordination strategy and coordination effectiveness. Furthermore, a coordination strategy is a group of coordination mechanisms. Coordination mechanisms are present to address dependencies in a situation. The conceptual framework is shown schematically in Figure 7.

Following Espinosa et al. (2004), and based on evidence from the literature review showing that coordination contributes to project success (as discussed in section 2.2.7 Project Success and Coordination), the conceptual framework also proposes that coordination effectiveness is an antecedent to project success. Coordination effectiveness is, however, only one of many factors contributing to project success. A project may be well coordinated yet be unsuccessful for reasons unrelated to coordination, such as misinterpretation of requirements, poor quality code, or 'political' interference.

This study explores the concepts of coordination strategy, coordination effectiveness, and the relationship between them in the context of agile software development projects. Since the focus is solely on coordination, project success and its other antecedents are outside the scope of the study.

Other factors are also of interest in this study of coordination. The literature review identified five factors influencing project coordination: team size, task interdependence, team expertise, project complexity, and project uncertainty. These factors have also been found to influence the success of agile software development projects, at least in practitioner literature. Therefore, complexity and uncertainty are incorporated into the conceptual framework based on the definitions provided by Xia and Lee (2005) (see Table 6) and Barki et al. (1993) (see Table 7). Xia and Lee (2005) define complexity using change and interdependency items, and Barki et al. (1993) define uncertainty using change, interdependency, expertise, complexity, and size variables. Therefore, expertise, change, and project size are implicit in the conceptual framework because they contribute to complexity and uncertainty. The conceptual framework proposes that complexity and uncertainty influence coordination in agile software development projects, however the exact nature of this influence is unclear.

Figure 7 The conceptual framework

The conceptual framework developed here, guides the research design, data collection, and data analysis by providing focusing concepts for the research.

2.4 THE RESEARCH QUESTIONS IN DETAIL

The preceding arguments lead to the formulation of two research questions. The purpose of the first question is to develop a well-informed set of concepts about coordination and dependencies in agile development projects, based on empirical data. The purpose of the second question is to explore the relationship between those concepts and the effectiveness of coordination within agile development projects. Each question is decomposed into sub-questions to provide more explicit information about the coordination of agile development projects. The first research question is:

RQ1. How is coordination achieved when using an agile development approach?

To identify how coordination is achieved in an agile software development project, it is necessary to identify all of the coordination mechanisms in use in a project. Coordination Theory indicates that coordination mechanisms only exist to address dependencies. In order to identify coordination mechanisms in a situation it is therefore necessary to identify the dependencies they address. Consequently, it is

necessary to know what dependencies are present within agile software development projects. This leads to the first two sub-questions:

RQ 1a What dependencies are present in an agile development approach?

RQ 1b What coordination mechanisms are in use in an agile development approach?

Agile approaches are achieved through the use of explicit practices from agile methods, and many of these practices are expected to be coordination mechanisms. The literature review discussed research supporting this conjecture (MacKenzie & Monk, 2004; Pikkarainen, et al., 2008; Sharp & Robinson, 2004). To identify those practices that provide coordination and are also agile practices, the third sub-question asks:

RQ 1c What agile development practices act as coordination mechanisms in agile software development projects?

Once dependencies and coordination mechanisms are identified, then the coordination strategy used in a project can be determined. To identify the coordination strategy, it is necessary to identify which coordination mechanisms manage which dependencies.

This will provide information on:

- which dependencies are not addressed (or are unmanaged) when using an agile development approach,
- which dependencies are addressed by multiple coordination mechanisms and why,
- which coordination mechanisms, that are agile practices, may be substituted for one another in a situation. This might be considered a form of flexibility,
- and, which coordination mechanisms, and therefore which agile practices address more than a single project dependency. This might be considered a form of efficiency.

This leads to the fourth sub-question:

RQ 1d How are coordination mechanisms related to dependencies in agile development projects?

Once the coordination strategy in an agile software development approach is identified then it becomes possible to explore the relationship between an agile coordination strategy and coordination effectiveness. This leads to the second research question:

RQ2. What is the relationship between the coordination strategy of an agile development approach and project coordination effectiveness?

Project coordination effectiveness is not defined in the research literature in agile software development, and therefore before research question 2 can be addressed in full, it is first necessary to define project coordination effectiveness in this context. This leads to the sub-question:

RQ 2a What is project coordination effectiveness in the context of agile software development?

Once a coordination effectiveness concept is defined, and an understanding of the coordination strategy in agile software development project is determined, then it becomes possible to gain an understanding about any differences between different coordination strategies, if any, and project coordination effectiveness. This leads to the sub-question:

RQ 2b How do different coordination strategies contribute to agile software project coordination effectiveness?

The literature review has shown that project complexity (Barki, et al., 2001), and uncertainty (Nidumolu, 1996) are important factors in studies of coordination and project effectiveness. They are also considered to be important factors in studies of software development (Little, 2005; Ratbe, King, & Kim, 2000; Xia & Lee, 2005). This leads to the following sub-question:

RQ 2c How do project complexity and project uncertainty influence the relationship between coordination strategy and coordination effectiveness?

2.5 SUMMARY

This literature review chapter has provided background to the proposed study of coordination within agile development projects, and developed a conceptual framework and research questions to guide the study. The review had two major strands, the first addressed agile software development and the second addressed coordination.

The first strand began with a brief history of system development methodologies, followed by a discussion to clarify the terminology confusion in that field. Then major reviews of agile software development literature were examined. This showed that research into agile approaches covers many topics but few in depth. Ethnographic

studies of agile approaches were reviewed because they provide details of how the approach works in practice. Research into iterative and incremental development was reviewed, and highlighted the benefits of the approach. Then literature focusing on how agile practices occur naturally under conditions of time pressure and uncertainty was reviewed, highlighting the difficulty of defining what 'agile software development' means. Research on tailoring of agile methods highlighted further difficulties in defining the approach. Then theoretical work defining agility as it relates to software development was reviewed. Finally, a summary of agile methods literature was provided, including information on coordination in agile software development.

Coordination was the second major strand of the review. The review covered coordination in organisation studies, IS project studies, and teamwork studies. Malone and Crowston's (1994) Coordination Theory was discussed in detail, and finally, the sparse literature using coordination to explain agile software development was reviewed. Then IS success, IS project management success, and IS project success literature was reviewed. This literature showed the positive relationship between project coordination and project success, and why project coordination effectiveness could be used as a criterion variable in this study. Project complexity and uncertainty were reviewed because IS literature and agile literature identifies them as important influences on software projects.

The final section of this chapter developed a conceptual framework proposing a relationship between an agile software development coordination strategy and project coordination effectiveness, influenced by project complexity and uncertainty. The conceptual framework led to two principle research questions, and a number of sub-questions.

The next chapter describes the research methodology for this study.

3. RESEARCH METHODOLOGY

A research methodology provides a systematic way to address research questions. A research methodology consists of a research strategy and a research design including methods for collecting and analysing data. This study uses the positivist multi-case study research strategy which is appropriate for building theory from field data collected from complex information system development projects (Eisenhardt & Graebner, 2007).

This chapter justifies and explains the research methodology, and has the following structure:

- Research paradigm and approach
- Overview of the research process
- Case design
 - Unit of analysis
 - Case selection
 - Data collection
 - Data analysis
 - Within case analysis
 - Cross-case analysis
- Ethical considerations
- Validity, reliability, and relevance
- The elements of theory
- Summary

3.1 RESEARCH PARADIGM AND APPROACH

A research methodology must be appropriate to address the research questions and normally aligns to an appropriate research paradigm. A research paradigm, or philosophy, is a basic belief system made up of consistent epistemological and ontological beliefs (Guba and Lincoln (1994). Epistemology is concerned with the nature of knowledge, what valid knowledge is, and how valid knowledge is acquired. Ontology is concerned with what entities exist, and the relationships among those entities. Understanding the research paradigm in a study is useful because it influences the choice of research methodology.

Within the information systems field there are three commonly accepted research paradigms: positivist, interpretive, and critical (Avison & Pries-Heje, 2005; Liu & Meyers, 2011, Orlikowski & Baroudi, 1991; Walsham, 2006). Positivism is the dominant paradigm in information systems research (Liu & Meyers). Epistemologically, positivism emulates natural science whereby researchers adopt a scientific and objective perspective, they assume valid knowledge is created from an understanding of phenomena and the relationships between phenomena, and that these relationships are repeatable and measurable. Researchers take the ontological stance that there is one knowable and observable reality. A research methodology aligned with the positivist paradigm focuses on developing and testing falsifiable propositions that specify relationships between concepts occurring in the world (Orlikowski & Baroudi, 1991).

The interpretive philosophy assumes that reality is subjective and formed by social interaction between people. Orlikowski and Baroudi (1991) explain it this way:

"The aim of all interpretive research is to understand how all members of a social group, through their participation in social processes, enact their particular realities and endow them with meaning, and to show how these meanings, beliefs, and intentions of the members help to constitute social action." (ibid, p. 13)

Epistemologically, interpretive research assumes that valid knowledge is a social construct arising from people's subjective and inter-subjective perceptions as they interact with one another. Researchers interpret what is occurring in a social situation with the assumption that there are multiple valid realities. Causality is explained as a cycle of reinforcing social interactions between actors. The research methods used are generally field studies examining humans in social settings, and research is designed to avoid imposing externally or predefined categories on the setting.

Critical research is common in social science but very uncommon in the field of information systems (Liu & Meyers, 2011). Critical research is concerned with critically evaluating social reality and the conflicts and contradictions therein, with the aim of transforming society for the better. Epistemologically, critical research assumes that social situations are artefacts of historical actions and beliefs that act to constrain people. Critical research focuses on understanding these artefacts and proposing change and improvement. The ontological stance of critical research is holistic; historical and contextual elements cannot be studied in isolation but must be studied as a whole. Furthermore, all social relations constantly undergo change, so critical research involves studying how historical and contextual factors change over time.

Critical studies tend to be longitudinal or ethnographic; involving studies of organisational processes and structures (Orlikowski & Baroudi, 1991). One difference between critical, interpretive, and positivist research is the use of theoretical frameworks to guide the study. Positivist and critical researchers enter the field with informing theory, whereas interpretive researchers aim to avoid preconceptions.

Paradigm choice is influenced by 1) the basic belief system of the researcher, 2) the research questions, and 3) the nature of the phenomenon of interest (Guba and Lincoln, 1994; Orlikowski & Baroudi, 1991). The positivist research paradigm most closely aligns with the basic belief system, or world view, of the researcher in this study. Otherwise, the research questions could legitimately be addressed within a positivist or an interpretive paradigm. Militating against an interpretive stance is the use of a conceptual model based on concepts from existing research, which is generally considered inappropriate in interpretive studies. The phenomena of interest, coordination practices in agile software development projects, could also have been studied from the positivist or interpretive perspective. The underlying conceptual model, however, focuses on coordination mechanisms forming a coordination strategy. A coordination strategy is not a social construct since it involves actual observable actions taken by software developers to organise their work. Since the focus of this research is not on peoples different interpretations of these mechanisms, but rather on overt actions, an interpretive approach might not have been so useful in addressing the research questions (e.g. RQ1c What agile development practices act as coordination mechanisms?). Coordination effectiveness is somewhat different, and might be more amenable to an interpretive stance, since different people might interpret effectiveness differently. Another consideration related to the paradigm employed in this study, is that the aim of the study is to develop a theory with falsifiable propositions (although testing those propositions is left for future work). Development of propositions is considered useful in positivist research, but not in interpretive research. Critical research is not considered appropriate for this research study because, although many people believe that agile software development improves the working conditions of software developers and the quality of software (e.g. Beck, 2000; Cockburn, 2002), thus ultimately contributing to societal improvement, this is not a focus in this study. Based on these considerations, a positivist stance is taken in this research to maintain a consistent and appropriate paradigmatic stance.

Qualitative Positivist Research

The discipline of information systems has a well-founded tradition of positivist research. Most research in the positivist tradition is quantitative and involves numbers and measurement (Straub, Gefen, & Boudreau, 2005), but a small body of positivist research is qualitative, involving words and other non-numeric data (Dube & Pare, 2003; Eisenhardt, 1989; Pare, 2004). These two forms of positivist research differ in their goals. The goal of quantitative positivist research is to generalise research findings to populations of interest, and data analysis is primarily deductive involving the testing of theoretical propositions developed *a priori*. In contrast, qualitative positivist research has the goal of generalising research findings to theoretical concepts of interest, and data analysis is primarily inductive involving building theory directly from empirical data. These distinctions are not absolute; studies can legitimately combine both quantitative and qualitative data, deductive and inductive analysis (Lee, 1991).

Positivist qualitative research and positivist quantitative research fit together in building and testing theory. Eisenhardt and Graebner (2007) explain that a positivist qualitative research methodology is appropriate to build theory by defining concepts or constructs (constructs are more precisely defined than concepts (Suddaby, 2010)), and propositions linking those concepts. Such theory is then testable using a positivist quantitative research methodology.

Justification for Qualitative Research

Although the positivist research paradigm underpins this research, a qualitative research methodology was selected to address the problem of understanding coordination in agile software development projects. Qualitative research is an appropriate choice when the existing theory on a topic is immature. Edmondson and McManus (2007) proposed this when they identified three archetypes of research theory maturity. Each archetype is based on the status of the underlying theory base, which can be mature, intermediate, or nascent. Mature theory has “*well-developed constructs and models that have been studied over time with increasing precision by a variety of scholars, resulting in a body of work consisting of points of broad agreement that represent cumulative knowledge gained*” (ibid. p. 1158). Nascent theory is at the other extreme and is characterised by the existence of little or no theory and the phenomena explored are often new. Such research “*proposes tentative answers to novel questions of how and why, often merely suggesting new connections among phenomena*”

(ibid, p. 1158). Intermediate theory fits between these extremes and is characterised by research that “*draws from prior work – often from separate bodies of literature - to propose new constructs and/or provisional theoretical relationships*” (ibid, p. 1165). Each archetype is associated with a particular research method, qualitative for nascent theory, quantitative for mature theory, and hybrid for an intermediate theory base. Hybrid methods mix qualitative and quantitative methods in some meaningful configuration (Gallivan, 1997; Miles & Huberman, 1994).

In this study, a mix of theory at different maturity levels informed the initial conceptual model. Coordination Theory (Crowston, et al., 2006) is at the intermediate level of theory. This theory is well conceptualised and has been employed to investigate the coordination aspects of numerous software engineering projects. This is not a theory for prediction, however, and does not enable measurement of constructs, so does not meet the criteria for a mature theory. Teamwork-based coordination theories can also be categorised at the intermediate level, because this research area has a multiplicity of constructs and no distinctive single theory exists (Mohammed, et al., 2010).

Coordination effectiveness literature also appears to have an intermediate theory-base, since the concept exists, but there is no consensus on a definition (Espinosa, et al., 2007; Faraj & Sproull, 2000; Kraut & Streeter, 1995).

The theory base for agile software development is nascent, or immature, as determined by Dyba and Dingsoyr (2008). Their systematic review of the literature on agile methods covered the period up until the end of 2005. Since then there have been studies using, and even extending theory on aspects of the approach, particularly using control theory (Harris, et al., 2009; Maruping, Venkatesh, & Agarwal, 2009). But overall, this research area is still largely uninformed by theory.

Based on these arguments, either a hybrid research design or a qualitative design seemed the best option for this study. Furthermore, the literature review showed that, although a small body of research identifies the presence of coordinative practices in agile software development projects, there is a lack of theory in this context. Therefore, a qualitative methodology was chosen for this study.

Qualitative Research Approaches

Having selected a positivist paradigm, and a qualitative research approach as appropriate for a study of coordination in agile software development projects, it was then necessary to choose a research methodology. According to Liu and Meyers (2012) the most common research methodologies used in the six highest-quality IS research

journals are survey, lab experiment, field experiment, case study and action research. Surveys and experiments tend to be quantitative in nature. This study required in-depth understanding of coordinative aspects of day-to-day work making quantitative methods less appropriate because detail is difficult to acquire using these methods. Also, as discussed in the previous section, the theory base of agile software development research is immature and more amenable to qualitative or hybrid methodologies. This leaves case study and action research as candidates for this study. Other, less commonly used qualitative methods could have been used including ethnography, grounded theory, or phenomenology; although they are not common in IS research. Phenomenology is a research approach performed within an interpretive paradigm, therefore it was not considered suitable for this study. Many other approaches exist but they are not all discussed here as they were not considered suitable for an explorative study of coordinative practices in ongoing software projects. They include: design science, simulation, theorem proof, argumentation, and literature survey. Somewhat more appropriate are the approaches provided by ethnography, grounded theory, and action research. Each of these is now considered.

Ethnography is a research method used in sociology where the researcher immerses themselves in a research site to gain exceptionally detailed understanding of a social situation. Research is most often longitudinal, and is not guided by pre-specified research questions. Ethnography was not considered appropriate for this research because ethnographic studies describe a situation in depth but do not propose theory or testable propositions, which is an aim of this study. The other reason was that ethnographic studies have already been used in this context, as discussed in the literature review of this thesis (e.g. Beynon-Davies, MacKay & Tudhope, 2000), and another ethnographic study of agile software development might not find anything new about coordination beyond what is already described in the extant literature.

Grounded theory is a research method used in sociology for developing theory about social situations (Glaser & Strauss, 1967). Formal grounded theory involves entering a field of inquiry without preconceptions, without performing prior extensive literature reviews, developing guiding conceptual frameworks, or defining research questions. The data collected may be of any type – qualitative or quantitative, although qualitative interviewing of participants is most typically used. Data is analysed using a constant comparative method of developing initial concepts and comparing those concepts with new concepts as new participants are interviewed or new data acquired. In this way, a theory emerges slowly as data is collected. Data analysis and data collection are

concurrent, and the final outcome is a core category explaining the concepts related to the phenomena of interest. Other outcomes may be research questions for further study, or even a theory with testable research propositions. Confusingly, sometimes in qualitative studies the grounded theory data analysis method of constant comparison is used in any situation where qualitative data analysis is required (Eisenhardt & Graebner, 2007). This is because the constant comparison technique is an inductive method useful for developing theory that closely fits the data. Thus the term grounded theory is used in two different ways by researchers; as a formally defined research approach, and also as a data analysis technique (Eisenhardt & Graebner, 2007).

Of particular importance in a formal grounded theory study is the concept of theoretical saturation which indicates to the researcher when data collection can stop. *"Saturation means that no additional data are being found whereby the sociologist can develop properties of the category."* (Glaser & Strauss, p. 61). At the point of theoretical saturation in a study, the researcher is confident that the theory and concepts contributing to the theory are valid.

The reason why formal grounded theory was not selected for this study is because of the large body of existing theory that was available to inform the study. This included prior research on coordination in general, and on coordination in large scale and distributed software development projects. This extant research literature could provide any number of useful research questions, conceptual frameworks, and informing literature to aid in design of the study. Choosing formal grounded theory might have limited the researcher in exploring this literature fully and using it to inform the study design.

Action research is used to gain knowledge about a phenomenon by intervening in a social situation to solve a practical problem. It stems from the field of education when new initiatives in education needed to be implemented and their impact studied in the field (Liu & Meyers, 2011). There was no proposed intervention in this study because the phenomena of interest was the existing practices in use, therefore this research approach was also not considered appropriate for the purposes of this study.

Case Study Research

The case study approach was chosen for this study. This approach is a comprehensive research strategy defined by Yin (2003b) as *"an empirical enquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident"* (Yin, 2003b, p. 13). The case

study research strategy is commonly used in information systems (Benbasat, Goldstein, & Mead, 1987; Cavaye, 1996; Dube & Pare, 2003), the related field of organisation studies (Eisenhardt, 1989), and social science in general (Yin, 2003b).

There were a number of reasons why case study research was considered appropriate for this study. Case study research can be carried out under the philosophical belief system of positivism. Case study research is well-accepted and understood in IS research which provides detailed guidelines on how to use the approach. Furthermore, there are a variety of high quality examples of the use of case study research in IS contexts. These points are now discussed in more detail.

Case study research can be carried out within any of the three research paradigms: positivist, interpretive, or critical (Cavaye, 1996). Positivist forms are currently the most popular, followed by interpretive, whereas there are few examples of critical case studies. This is what Dubé and Paré (2003) found when they surveyed seven high-ranked IS journals from 1990 to 1999. Positivist case study research was used in approximately 13% of the articles. A more recent survey, by Liu and Meyers (2011) of the Association for Information Systems six highest-ranked IS journals, covering the period 1998 to 2007, found positivist case study research was used in at least 19% of all research articles.

In IS, there are clear guidelines for performing positivist case research (Cavaye, 1996; Darke, Shanks, & Broadbent, 1998; Dube & Pare, 2003; Eisenhardt, 1989; Lee, 1989; Pare, 2004). These guidelines all draw in part from the early work of Yin (1984), who discussed how the tenets of logical positivism can be applied in case study design, data collection, and data analysis. Eisenhardt (1989) provides steps to follow when building theory from positivist multi-case study research. Her 'roadmap' draws on guidelines from Yin (1984) on case design, Miles and Huberman (1994) on data analysis techniques, and Glaser and Strauss (1967) on grounded theory. She also includes guidance based on her own experience and insights, in particular, the importance and use of *a priori* theory, the cross-case analysis technique, and how extant literature reinforces emergent theory. She also recommends between 4 and 10 cases in a multi case study to provide a balance of validity and manageability. Paré (2004) provides similar detailed guidance on improving the rigor of information systems research when using the positivist theory-building case study approach.

The field of information systems provides clear guidelines for performing positivist cases studies, along with exemplars. Exemplars cover both system development, and

implementation. For example, Fitzgerald et al. (2006) used a single in-depth positivist case study to explore how Scrum and XP are complementary, and can be successfully combined to address management and technical aspects of development. Paré (2004) provided an analysis of two exemplars when he compared two case studies to illustrate issues of rigor in exploratory theory-building case studies. One is a single case study by Keil (1995) exploring troubled IT projects, and the other a multi-case reported by Paré and Elam (1997) exploring factors influencing project implementation success.

Although guidelines and exemplars of positivist case study research are readily available, there were further reasons for choosing the case study strategy for this study. These reasons include the nature of software development projects, the need for a theory building method, and the ability to perform multi-case research.

The nature of software development projects is messy; they are complex and involve multiple interacting factors. To explore coordination it was going to be necessary to carry out an in-depth and detailed exploration within current and ongoing agile software development projects. An in depth study was necessary because each project was likely to be using a slightly different development method, and therefore a variety of different coordination mechanisms to address project dependencies were likely to be present. In addition, different organisational, social, technical, and process factors were likely to be occurring in each project. Case study research enables in-depth study in situations such as these (Benbasat, et al., 1987; Cavaye, 1996; Darke, et al., 1998; Yin, 2003a). Case study research achieves depth by supporting the collection and analysis of a rich variety of data. Data can come from people, in the form of interviews, observations, and surveys, from documents, and from other project artefacts, for example, 'story cards', 'burn-down charts', and photographs of the work site. A range of data of this type was considered necessary to provide detailed evidence showing how coordination is achieved in a project.

Another reason for choosing the case study strategy is its usefulness as a tool for theory building. Eisenhardt (1989) proposes that case study research can generate well-conceptualised theory based on real, as opposed to theoretical, situations. Theory building positivist case studies are generally exploratory in nature, and Paré (2004) tells us that, "*an exploratory case study, whether based on single or multiple cases, is aimed at defining questions, constructs, propositions, or hypotheses to be the object of subsequent empirical study*" (Pare, 2004, p. 235). The aim of this study is to generate theory about the concepts of coordination strategy and coordination effectiveness, and

to define propositions describing the relationship between them. This theory had to be in a form suitable for future empirical testing using quantitative methods. The case study approach supports this type of theory building.

Choosing the case study research approach also allows for a multiple case design. A single case design is appropriate when a case is unique or revelatory (Yin, 2003b). Agile software development projects are now common, making the selection of multiple cases possible. A multi-case study enhances the external validity of findings by showing that the theory generated is applicable in more than a single instance. The rationale for case selection is explained in the case design section of this chapter.

3.2 OVERVIEW OF THE RESEARCH PROCESS

The research process follows a sequence designed to improve the rigor of IS case study research developed by Dubé and Paré (2003). Their guidelines cover each phase of the process, the research design, data collection, and analysis of the data.

The following steps summarise the overall research process followed in this study.

1. A review of pertinent literature was carried out. This included research on agile software development and coordination.
2. Based on the literature review findings, an initial conceptual model was developed to provide high-level concepts and their relationships to guide the research design, data collection, and analysis. This framework also formed the basis of two principle research questions and sub-questions.
3. Theoretical and empirical literature was located to verify that a positivist qualitative research paradigm was appropriate to address the research questions.
4. The unit of analysis was defined based on the research questions. The unit was *a co-located agile software development project*.
5. Case selection criteria were determined based on theoretical and practical criteria and an appropriate number of cases was determined. Three typical agile projects and one non-agile project were chosen as a minimum number to allow for theory development.
6. A case study protocol based on the research questions, conceptual model, and unit of analysis was designed (Yin, 2003b). This protocol included an interview protocol, and the nature of appropriate additional data sources. In addition, a

participant information sheet, and a protocol for conduct at the case site were developed.

7. A pilot case study was conducted to assess the case study protocol, and verify it was adequate to address the research questions. Data were collected at the pilot site and minor adjustments were made to the protocol before the main cases were selected.
8. Four suitable cases were located, each from a different organisation. Data were collected from these cases.
9. Data analysis was performed on each case individually. For each case:
 - a. A full case description of each project was written based on a framework written for the purpose of clarifying information about the organisation, the project, project stakeholders, the team, the development method, and major project issues.
 - b. The project coordination strategy was determined.
10. Cross-case analysis was performed by amalgamating findings from within-case data. These steps were followed:
 - a. A generic concept of 'coordination strategy' was developed by amalgamating the models from each individual case.
 - b. A definition of 'coordination effectiveness' was developed from data amalgamated from all cases.
 - c. Relationships between these concepts were developed.
 - d. Propositions were written proposing the relationship between coordination strategy and project coordination effectiveness.

Figure 8 illustrates this research process and the aim of each step.

Figure 8 The research process

3.3 CASE DESIGN

Case study design involves identifying the unit of analysis, choosing theoretical and practical criteria for case selection, determining an appropriate number of cases, selecting appropriate methods for collecting data pertaining to each case, and designing data collection instruments. The data analysis phase also involves design decisions. For example, should a pre-defined framework of categories or analytic codes guide data analysis, or should the analysis be purely inductive? The following sections describe how this study addresses these issues.

3.3.1 UNIT OF ANALYSIS

In case study research, the unit of analysis determines the ‘case’ under investigation (Yin, 2003), and sets a boundary around the phenomenon of interest (Miles & Huberman, 1994). The unit of analysis can be any bounded phenomenon such as an individual, dyad, group, role, process, project, activity, event, organisation, nation, intervention, or geographical location (Miles & Huberman, 1994). In this study, the most appropriate units in a study of coordination were considered to be the group (i.e. the software team), the process or activity (i.e. agile software development), or a temporally bounded event (i.e. the project). Team was not selected because this is not a study of teamwork, but rather a study of coordination, which is a broader concept involving practices, artefacts, and technologies used, as well as the interactions between people. The process of agile software development was not selected because this study is not just a study of the agile process, but of all coordinative activities and artifacts involved in an agile project. Following the advice of Miles and Huberman (1994) on case selection, the unit of analysis in this study was primarily determined by the research question, which is a study of projects. More specifically, the unit of analysis is *an agile software development project with co-located participants*. Co-location was included in the specification of the unit of analysis because co-location is the more usual state for an agile software project (Beck, 2000; Schwaber & Beedle, 2002).

This unit of analysis provides a number of benefits. Most importantly, it enables the research questions to be addressed while simplifying theoretical case selection because it is easy to select cases based on project characteristics such as team size, and relative problem complexity. Furthermore, because practitioners find the concept of ‘agile software development project’ easy to understand, it simplifies discussions about

suitable cases. A project has certain simple characteristics: a start date, an end date, a reasonably unchanging group of participants, a predefined system development method, and participants work on the same problem, which is to create a software system to meet a business purpose determined by some external party. These project characteristics provided clear bounds on each case. With a project as the 'case', it is simple to work out a suitable time-period for collecting meaningful data. In addition, this unit simplifies decisions about participant selection, as only major external project stakeholders, and those involved in the day-to-day work of the project are of interest.

3.3.2 CASE SELECTION

Theoretical criteria and practical constraints guide case selection. Theoretical criteria ensure that concepts and relationships in the initial conceptual model are explored, whereas practical considerations constrain the nature and number of cases.

Theoretical Case Selection Criteria

Theory-based case selection is a recommended practice in multi-case study research when working in the positivist paradigm. Eisenhardt (1989), Miles and Huberman (1994), Paré (2004), and Yin (2003b) advocate for pre-designing the number of cases based on theoretical considerations. This is achieved with replication logic which involves selecting cases because they will tend to either confirm or disconfirm the theory or propositions of the study (Yin, 2003b). Confirming cases are selected based on literal replication, that is, similar results are likely. Disconfirming cases are based on theoretical replication, that is, contrasting results are likely, but for predictable reasons. Yin (2003b) likens this selection logic to that of traditional experimental design where experiments are either repeated to confirm a result or, in other instances, one experimental variable is changed and the experiment repeated to see the affect of the changed variable on the result.

Literal replication cases met the following theoretical criteria:

- The project was using a published and commonly known agile method such as Scrum or Extreme Programming.
- The project consisted of an identifiable group of participants, or team.
- The project team was co-located.
- The use of the agile method was reasonably mature, in that most of the project team members had worked this way before. Although it would have been optimal to select projects with participants who had experienced many

successfully completed agile projects, it was not possible to locate such projects, so the criterion was relaxed to accept projects that were near completion and had successfully completed a number of iterations.

- The project had a clear business purpose and provided a tangible business benefit to the organisation. All projects were assessed to ensure that the project product (the software) was created for the use of an external party and was not just for the use of the team itself. In other words, projects were either producing software for a client organisation, or for another unit within the same organisation.
- The project had 2 to 10 directly involved participants. A team of 10 is the maximum size considered suitable for agile projects (Beck, 2000; Schwaber & Beedle, 2002).
- Projects varied along the dimensions of project complexity and project uncertainty.

Practical Case Selection Criteria

Eisenhardt (1989) argues that the number of cases in theory-building from positivist case study research should be between 4 and 10 for the practical reasons of controlling time spent in the field and managing complexity. Achieving theoretical saturation might be a preferred way to decide when to stop selecting new cases to study, and this was discussed in the section on grounded theory, but with each new case added to a study, considerable time and effort is needed to find a case, collect data, and analyse that data. Eisenhardt (1989) argued that four cases provide enough evidence to achieve a convincing theory with adequate empirical grounding. More than 10 cases increases complexity and is probably beyond the ability of a single researcher to manage. Furthermore, the within-case analysis might lack adequate depth. Based on this advice, in this study the researcher chose four cases to ensure manageability, to provide enough cases to support the theoretical design discussed in the previous section, and to provide enough data for meaningful theory development.

One practical constraint on case selection was that cases should be located in Wellington, the capital city of New Zealand. This eliminated the need for the researcher to travel within New Zealand or overseas. This constraint did not limit the study to one cultural milieu as software teams in Wellington are multi-cultural due to the low availability of local expertise in software development, and the high desirability of New Zealand as a place to live. In addition, selecting Wellington as the location for projects

provided a wide range of possible projects. Wellington is the location of many small to medium-sized IT companies, many company Head Offices, and all New Zealand Government department head offices.

In addition, all projects met the following practical criteria:

- Formal approval was granted by the organisation where the project was carried out.
- Entry to the work place was permitted. This ensured that visual assessment of the work site layout, and public artefacts such as wallboards could take place because these can be a factors influencing coordination.
- At least one person associated with the project (typically a project manager, team leader, senior business analyst, or senior developer) was available and prepared to provide a comprehensive overview of the project, its purpose, history, development method, and major issues, its importance to the organisation, and its perceived success.
- Between two and four other team members were available for interviewing. This included customers (or clients, or end-users), if they were considered part of the team. The minimum requirement was to interview at least one senior team member and one developer. Selecting at least one software developer was an important consideration as they work at the 'heart' of the project and would have in-depth experience with its coordination practices.
- The project was well underway in that it had progressed through at least half of its estimated duration. This meant that at the time of the study, the project had a distinct start date, and an end date, or projected end-date. This also ensured that enough of the project was complete for the participants to comment meaningfully on the project and its coordination.
- If the project was complete, it had finished within the previous month. This was to ensure that participants were recalling recent events.

Identifying Candidate Projects

Cases meeting the theoretical and practical criteria of the study were located through the researcher's professional network. For example, initial contacts were made through the local Agile Professionals Network (APN⁸), a not-for-profit organisation that focuses

⁸ <http://www.agileprofessionals.net/>

on informing the local software project community about agile software development. This organisation held regular seminars with invited practitioners as speakers, and some of these speakers were approached to discuss candidate projects. Other sources included organisations and people identified in local trade publications, who had discussed their adoption of agile software development methods. All of these sources provided organisations or people who might have projects that met some of the selection criteria. After identifying candidates to approach, a meeting was initiated with one person working in an organisation hosting a candidate project. This contact was by personal introduction. For example, the researcher would ask a contact to introduce her, in person or by email, to someone working within the organisation of interest. Then an informal meeting, called a 'kick-off' meeting, was set up with the organisational contact to discuss the profile of the project required, and ask if their organisation had any suitable projects that met the practical and theoretical criteria of the study. If they did have a suitable project, then the researcher asked if the organisation would be willing to offer that project as a case in the study.

Some projects were rejected at this stage, either they did not meet the selection criteria, or the initial contact person would approach the organisation, and the people within the organisation were not prepared to proceed. Following this process, five suitable cases were selected for the study.

Overview of Case Selection

The first case was a pilot case. The data from this case did not contribute to the findings. The purpose of the pilot case was to confirm the case protocol, logistics, data gathering procedures, and instruments, were fit for their purpose. Selection of the pilot case was for convenience rather than to meet any specific criteria, and data collected were not analysed.

The four cases contributing to the findings consisted of three literal replications and one theoretical replication (Yin, 2003b). Literal replication logic, when cases are similar and are expected to provide similar results, meant selecting cases using agile software development practices, whereas theoretical replication was achieved by selecting a single non-agile case.

Because of the single-researcher constraint, case selection and data collection were conducted sequentially. The plan was to select four independent cases, which was deemed a suitable number given the single researcher and time constraints on the research (Eisenhardt, 1989). Initially, three independent replication projects were selected, each from a different organisation. Although these cases were similar, because they were all using the Scrum methodology or Scrum with XP practices, they varied across the factors of interest in the study; in particular, they varied in terms of their relative project complexity, and uncertainty. Project complexity and uncertainty were informally assessed in initial discussions with the key research informant on each project. Three cases were identified, and interviews carried out over the period from 25 November 2009 to 24 March 2010. These cases were code-named Land, Storm, and Silver. Then some initial data analysis was performed. While this data analysis was underway, a further non-agile case was sought. This fourth case was named Rock. Data collection for this case was carried out from 18th April until the 4th May 2011.

Although three replication cases were selected, there was considerable variation between them. Projects were from different organisation types. Project Land occurred in a government department. Storm occurred in a quasi-public sector organisation (a State-Owned Enterprise⁹). Silver occurred in a small commercial software development house, and project Rock occurred in a commercial bank. A summary of the cases, their selection criteria, and the purpose of each case is provided in Table 8.

⁹ SOE: "SOEs are businesses (typically companies) listed in the First Schedule to the State-Owned Enterprises Act 1986. SOEs operate as a commercial business 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 Apr 2011. <http://www.treasury.govt.nz/glossary/soe>

Table 8 Cases and selection criteria

Code Name	Replication Type	Selection Criteria	Purpose and notes
Hour	Pilot case	Convenience of access and location	To confirm the case study protocol The project was not using an agile method
Land	Literal	Agile project using Scrum	To develop a tentative theory of coordination strategy and coordination effectiveness
Storm	Literal	Agile project using Scrum and some XP practices	To extend the tentative theory in another context An exemplary project because it had 10 developers, used all of the Scrum practices
Silver	Literal	Agile project using Scrum and some XP practices	To extend the tentative theory in another context
Rock	Theoretical	Non-agile project	To ensure the theory is relevant to agile software development and not to all types of software development

3.4 DATA COLLECTION

Data collection involved visiting project sites and collecting data from a variety of sources. Collecting data from different sources makes data triangulation possible, which is a recommended practice in case research.

Triangulation

Triangulation is important in case study research because it improves the validity of a study by providing corroboration of findings (validity is discussed further in the final section of this chapter). Yin (2003) discusses four common forms of triangulation pertinent in case study research:

1. Data triangulation – this is the use of different data sources (e.g. interviewing participants, observing participants, analysing documents, collecting photographic evidence)
2. Investigator triangulation – this is the use of more than one researcher in collecting and analysing case data
3. Theory triangulation – this is the use of different perspectives or conceptual frameworks when collecting and analysing data (e.g. theory of reasoned action, complexity theory)
4. Methodological triangulation – this is the use of different research methodologies in the collection and analysis of data (e.g. qualitative research methods, quantitative research methods)

Data triangulation improves the validity of findings because evidence for a finding comes from multiple sources. Data triangulation was used in this study. Investigator, theory, and methodology triangulation were not used due to limitations on thesis size, time available, and the requirement for a single researcher to carry out the research.

There are two ways to achieve data triangulation. One involves collecting data on the same topic from different sources of the same type. For example, the researcher collects data on the same topic asking the same question of two or more people. The other form of data triangulation is when data is collected from different types of data source. For example, the researcher collects data on the same topic by interviewing people and corroborates their reports with photographic or documentary evidence.

In this study, both of these data triangulation practices were used. The primary method of data collection was the semi-structured interview (that is, the interview followed a pre-designed schedule and included closed and open-ended questions). Various people working on the project were interviewed. Their selection is discussed in the next section.

Multiple different data sources were also used in this study. This meant that a wide range of data was collected; improving validity and also providing additional information that might not be evident if data came only from interviews. In this study, the data sources include interviews, observation with field notes written to describe project activities, questionnaires, photographs, and publicly available data from each organisation's web site, system documentation, and project documentation. All data were captured in electronic or paper form, and formed the database of case evidence for the project.

Participant Selection

Cases were selected that had a maximum of 10 people directly involved in the day-to-day work of the project. On each project, a maximum of five people were selected to be interviewed. This limit on the number of interviews was made for a number of reasons.

The first reason for choosing this limit was that it allowed for at least half of the project participants to be interviewed in each case. In projects with less than 10 project team members this would mean it might be possible to interview more than half of the people on the project.

The second reason for choosing this limit was to ensure the research could achieve adequate data collection and data analysis of each of the four planned cases in the time available.

The third reason for choosing this limit was that all interviews for a project could be carried out over a short period of one or two weeks. This was to try and ensure that all people interviewed were recalling events in the project over a fairly consistent time period, thus improving triangulation of their responses. Increasing the number of interviews per project would mean the total interviewing period for a case would be extended, and could lead to different participants reporting on different time periods in the project, this might have affected the usefulness of this aspect of triangulation.

Limiting the number of interviews in this way meant it was possible that less information on the factors of interest in the study would be collected. Limiting data collection was a practical compromise made to achieve maximum data from the available project team members while meeting restrictions on time in the field, and achieving a reasonable level of triangulation of responses.

Another criterion for selecting interviewees was based on their role in the project. People were selected who took a variety of roles in the project. This was done to ensure a variety of different perspectives on the coordination and other relevant events, within the project. For example, if a project team had five developers, one coach, one tester, and one designer, then the selection was based on role. The coach, tester, and designer would be interviewed along with two developers. Another criterion was that at least one developer was interviewed on each project. This decision was based on the idea that the developer was the person most closely involved in the day-to-day development work of the project and could offer insights into technical aspects of the project that might prove of interest in a study of coordination.

The order of interviewing was also pre-planned. The first person to be interviewed on a project was selected because they could give an in-depth overview of the project, its purpose, background, the roles of people on the project, and any important historical events that affected the project. That person could also assist in selecting the other people on the project who would be asked for interviews.

These following sub-sections explain the development of the interview protocol; describe the purpose of the other data sources, and the overall data collection process.

Interview Protocol

Interviews were the main mechanism for data collection. The interviews were crucial to the success of the study because they linked the research questions and initial theoretical concepts with the data, findings, and final theory.

To create an interview protocol, the literature on coordination in software development was consulted to determine the types of questions used in other studies of coordination mechanisms and dependencies. Two sources were particularly useful. Crowston's (1991) doctoral thesis provided an interview protocol focusing on dependencies and coordination mechanisms, when studying coordination in the change process of organisations. McChesney and Gallagher's (2004) journal article provided an interview protocol focusing on getting a project overview, in addition to aspects of project coordination, when studying coordination in software development projects.

Following Crowston (1991), another resource used in developing the interview protocol was a book by Spradley (1979). This book provides detailed advice on interviewing people, and how to draw out their experiences and perceptions of a situation, while avoiding 'leading the witness', or polluting the responses because people tend to try and give the 'expected' or 'right' answer. This was envisaged as a likely problem in the interviews because software developers may assume there is a 'textbook' answer. This is because the 'correct' way to adopt an agile method comes from specific books, or is taught using formal training sessions run by certified 'agile coaches'. Individuals are aware that they may not follow the recommended practices at all times. On Spradley's (1979) advice, 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?*"

The interview protocols used for the four post-pilot cases are provided in Appendix D Interview Protocols. Two protocols were developed. The first was primarily for collecting information about the project purpose, history, and status. This protocol was used when interviewing the first interviewee for a project, the project leader. This person was identified at the initial kick-off meeting during discussion with the key organisational contact. This interviewee was selected because they were able to give a full project description. The protocol had sections for collecting data about:

- Interviewee details including the name, role, and main goals of the person's work

- Organisation details, including the size and structure of the organisation
- Project profile and process, including stakeholder identification and communication, and details about the iterations and the activities carried out within iterations
- Coordination on the project covering factors contributing to coordination and hindering coordination
- The interviewee's ideas about what makes a project well-coordinated

The second schedule was for collecting information about coordination mechanisms and dependencies in the project. This involved eliciting details about the daily work of the team member. This schedule was used to collect interview data from up to four selected team members on a project. This schedule had sections for collecting data about:

- Interviewee details (as above)
- Daily work activities
- Coordination on the project (as above)
- The interviewee's ideas about coordination (as above)

The protocol was reviewed by two academics with experience in software development before it was used in the pilot case study. After the pilot study, a few minor changes were made to the questions, primarily to the order of the questions, to simplify the structure of the protocol.

Observation

Observation at the project work site was used to identify possible coordination mechanisms and dependencies in the project. This involved two different activities. First, viewing the work site and observing staff interactions and activities provided evidence of co-location, visibility of wallboards, and other displayed signage relevant to the project. Although all work sites were observed, staff were not always seen carrying out work. This happened when interviews occurred over the lunch break, if the project was in hiatus (i.e. Silver had a break in progress at one point), or if the project was complete (i.e. Rock). The researcher sketched the work site layout for all projects. Another form of observation occurred when attending stand-up meetings, or breakdown sessions. Field notes were made during, or immediately after, attending these sessions.

Questionnaires

Two questionnaires were developed and used to collect data on the project background and the agile practices used on the project. Their purpose was to collect project data efficiently and to save time during the interviews, rather than for statistical purposes. These questionnaires are provided in Appendix E The Questionnaires.

The first questionnaire was called 'Project Profile and Software Development Practices Questionnaire'. This questionnaire was completed by the project leader. It was used to gather information about the organisation, the problem addressed, the technologies, and the software development practices used on the project. The second questionnaire was called the 'Software Development Practices Questionnaire'. This questionnaire was a subset of the first questionnaire in that it only contained questions about the software development practices used on the project. One team member who was a software developer completed this second questionnaire. The purpose of this questionnaire to get another perspective on the agile practices used in the project, as developers and project leaders often have a different experience or understanding about certain practices, particularly if the project leader has no software development experience or is not intimately involved in the day-to-day project work. The questionnaire was closely based on that developed by Strode (2005), and included items on known practices from the five agile software development methods including Extreme Programming, Dynamic Systems Development Method, Adaptive Software Development, Scrum, and Crystal Clear.

The reason for collecting a list of agile practices was to gain a complete picture of the agile practices used in each project. Some of these practices were also identified in interviews but not all, because many of the practices are not coordination mechanisms.

Photographs

Photographs of publicly visible artefacts such as wallboards, burn down charts, done lists, and other publicly displayed artefacts relevant to the project were obtained whenever possible. In the completed project Rock, this was not possible, and in Land, there were no such artefacts. In Storm and Silver, the project leader was asked for permission before photographs were taken. In some cases the project leader took the photographs, in other cases the project leader provided existing photographs taken earlier in the project. Photographs were taken as evidence of the use of certain practices.

Web Site

The official organisation web site provided information about the organisation carrying out the project. These sites typically described the main business function of the organisation, its size, and structure. In some organisations, the latest annual report or other useful documents provided background information about the organisation, its projects, and their importance to the organisation.

Documentation

A variety of different kinds of document were collected, but not all projects could provide documents. Items such as story cards, Kanban cards, specification documents, and Rally™ product backlogs were collected. These documents, like the photographs, provided supporting evidence of the use of certain practices.

Sketches

Sketches of the room layout, and the position of the desks in the room were made for all on-going projects. This provided documented evidence of co-location and the visibility of any publicly visible artefacts such as wallboards or notices such as 'done' lists. These sketches were made either during or immediately after the site visit, and formed part of the case field notes.

On some projects, the participants made sketches and explanatory diagrams during interviews. These were usually architecture models or diagrams describing stakeholder types or wallboard layouts, and they too formed part of the case field notes.

Data Collection Process

After the kick off meeting, the initial contact introduced the researcher, in person or by email, to a person on the project who could provide a complete project overview. For the purposes of this study, this person was called the 'project leader' although these people fulfilled a variety of roles in the various projects.

The first formal interview was always with the project leader to collect data on the project purpose and history. Before the interview the project leader was sent a copy of the participant information sheet, an organisation consent form, a participant consent form, and the questionnaire, 'Project Profile and Software Development Practices Questionnaire' (see Appendices for these documents). The project leader interview process was as follows.

- Review the participant information sheet with the participant.

- Ask the project leader to complete and sign the project consent form and the personal consent form.
- If permission was granted, the digital recorder was switched on. Notes were taken during the interview in case the recorder failed.
- Review the responses to the 'Project Profile and Software Development Practices Questionnaire' and discuss any missing details or questions the participant did not understand. Missing responses were then completed on the document.
- Commence the interview following the 'project manager' interview schedule. This schedule was followed closely unless the participant answered questions out of turn, or described events that needed to be probed for further detail, or if the researcher decided to ask for examples to clarify an answer. In these instances the schedule was not followed exactly.
- Turn off the digital recorder.
- Discuss appropriate team members to interview and how best to approach them. In most cases, the project leader agreed to ask the team members if they would like to be interviewed, and collected the names of those who agreed, and then emailed the names to the researcher. As discussed above, the limit on the interviews in each project was five. In small teams, everyone on the team was interviewed. In larger teams, participants were selected based firstly on their role, but when there was more than one person taking a role, then they were selected on availability (i.e. the worked on-site most days), their length of time working on the project, and the likelihood that they would contribute willingly and fully in an interview situation. If the project was ongoing the researcher would ask if she could view the work site and attend a daily stand-up meeting, and if she could take photographs of wall boards and other artefacts that were publicly displayed for team use. The researcher also requested copies of any relevant documents such as architecture models, user stories, task cards, or other project documents mentioned at the interview.

If, during these interviews, an invitation was made to attend any meetings, such as a daily stand-up, then the researcher would attend and ask to be introduced to the team. After this meeting, each team member identified as a prospective interviewee was contacted by email, and a time, date, and location for an interview was arranged to suit their schedule. These people were sent the participant information sheet and the

participant consent form. One developer on the project team was sent the Project Profile and Software Development Practices Questionnaire.

All interviews were voluntary, and every person asked agreed to an interview. Every effort was made to suit the participant's timetable when organising the interviews. Interviewees were asked to organise a meeting place, which was sometimes at their workplace in a meeting room, sometimes at the researcher's university in a meeting room, and once at a local coffee shop. Team member interviews followed this process:

- Review the participant information sheet with the participant and show them the signed organisation consent form.
- Ask the participant to complete and sign the personal consent form.
- If permission was granted, the digital recorder was switched on. Notes were also taken during the interview.
- If the participant had completed the 'Software Development Practices Questionnaire' then this was briefly reviewed and any missing details or questions the participant did not understand were discussed and then completed.
- Commence the interview following the 'team member' interview schedule following the same process as for the project leader.
- Turn off the digital recorder.
- The researcher requested copies of any relevant documents such as architecture models, user stories, task cards, or other project documents mentioned at the interview.

Joint interviews were proposed if the project leader thought that some team members would be too shy to be interviewed alone. Only one interview was a joint interview. This interview was very successful because the participants were clearly good friends and explained or elaborated on each other's answers, reminded each other of missing details, and seemed relaxed during the interview.

All interviews took between one and 1 ½ hours. After the interview, a thank you email was sent to the participant. When all interviews in a project were complete, a small gift was given to the team to share.

3.5 DATA ANALYSIS

The purpose of data analysis was to find empirical evidence to build a theory about coordination in agile software development projects. Analysis involved two phases, within-case analysis, whereby each case was analysed independent of the others, and cross-case analysis to identify commonalities and differences between cases, and build the theory of coordination. Before analysis began, it was necessary to prepare the data for analysis.

Data preparation

The primary data source was the audio-recorded interview. The researcher transcribed a sample of five audio files and a professional transcription service transcribed the remainder. Data were transcribed into a template so that the final transcripts had the same structure. The researcher checked the professionally transcribed interviews against the recordings to ensure their accuracy. Once complete and checked, the transcripts were uploaded to the qualitative data analysis tool NVivo™.

Data from all sources were organised by case. For each case, an individual computer folder stored electronic records including audio files, transcripts, photographs, and field notes. Hard copy was kept whenever possible.

3.5.1 WITHIN-CASE ANALYSIS

Within-case analysis treats each case as an independent unit. Each case was analysed in two steps:

- A project description was written. This is a detailed description of the project recreated from the project data using a case description framework
- A project analysis was written based on a detailed analysis of each project data source. This analysis identified dependencies and their associated coordination mechanisms, and the overall coordination strategy of the project

The development of each project description and the analysis of each project's coordination strategy followed a coding procedure for qualitative data analysis described by Miles and Huberman (1994) and Thomas (2006). Coding involves identifying data items. A data item is a section of conversation, or other data such as a photograph, a section of project documentation, or a section from field notes, that share a common meaning. That meaning is given a descriptive name, called a 'code', which is

a label pertinent to the objective of the study. Each code is defined uniquely with respect to all other codes in the study.

Case Description Framework

Eisenhardt (1989) recommends the first phase of data analysis should be to describe each case in detail so the researcher becomes familiar with the case as an independent unit. To facilitate case description a case description framework was developed. The purpose of this framework was to familiarise the researcher with the case detail, and to reconstruct a coherent and organised version of case events and activities suitable for selected participants to review. This structured description also helped in organising evidence for later cross-case comparison. The framework consisted of attributes appropriate for describing a software development project. Attributes were selected from the literature, particularly from McChesney et al. (2004), the researcher's experience, and some attributes were developed inductively when analysing the first case, Land. The framework attributes are explained in Table 9.

The process for writing the case description was to enter the framework attributes into the InVivo™ application as codes. Each transcript was reviewed and transcript snippets were entered under the appropriate code. Once the coding was complete, then data attributed to each code was organised into a case description.

Analysis of Coordination Strategy

A coordination strategy is a group of coordination mechanisms used in a situation. Mechanisms form a strategy when they are selected consciously by project stakeholders, rather than occurring by chance. To identify the coordination strategy of a project, a detailed analysis of dependencies and coordination mechanisms was performed by analysing the transcripts, and all other relevant project data. The purpose of identifying dependencies is to isolate the coordination mechanisms in a situation, making it possible to build an understanding of the coordination strategy used in the project. Malone and Crowston (1994) proposed that a coordination mechanism only exists to manage a dependency, and a dependency forms when an activity is constrained in some way by another activity or the availability of a resource. Dependencies are 'managed' or 'unmanaged'. Managed dependencies are those with an identifiable coordination mechanism whereas an unmanaged dependency has no identifiable coordination mechanism, and therefore impedes project progress.

Table 9 Case description framework

Data sources	A list or table of data sources, their assigned identification code, and for interview data, the role of each research participant
Overview	An introduction to the organisation and the project
The organisation	<p>Organisation type Government, semi-autonomous government body, commercial, not-for-profit</p> <p>Organisation size Number of employees within New Zealand (approximate) and overseas</p> <p>Business activity The primary business activity of the organisation where the project was carried out</p> <p>Market The market the software product is be sold into</p> <p>Number of staff Number of staff involved in software development (approximate)</p> <p>IT Unit structure The structure of IT support and software development sections in the organisation</p>
The project	<p>Project status The completion state of the project when data collection occurred. Length of time the project had run including details of pre-project phases, the project start date, and end date</p> <p>Project purpose The aim of the project, what the project involved, including the business process the software was to support.</p> <p>Project criticality How important was this project to the organisations involved, to the participants, or to other stakeholders? How committed was everyone to the project? Was the project under time pressure or other constraints?</p> <p>Project history Any background information that influenced the current project status</p> <p>Project stakeholders The role and primary job activity of each team member. The internal and external stakeholders and their main role</p> <p>Perceptions of project success What the team or any other stakeholders said about the project success</p>
The technology	<p>Tools The software programming languages and platforms used to develop the software product. Any other software applications used to support development</p> <p>Architecture The component parts of the software product and any other systems integrated with or linked to that product</p>
The team	<p>Team location How the team were located with respect to each other. Room arrangements</p> <p>Length of time working together How long the team members had worked together</p>
The development method	The system development methodology used in the project, including the number of agile projects completed by the team, the previous experience of agile methods of individual team members, any history of changes in the method or individual practices, and the name and a description of each development practice mentioned in the interviews
Project issues	Issues raised by the participants that had an impact on project progress

Crowston and Osborne's (2003) heuristics for identifying coordination mechanisms and dependencies in the data were followed:

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 & Osborn, 2003, p. 352)*

Coordination mechanisms and dependencies were identified and then coded using the NVivo™ tool. Initially a predetermined set of codes was used. These codes were based on an analysis of the expected coordination mechanisms present in the agile methods XP, Scrum, and DSDM, and from dependencies and coordination mechanisms identified in the literature review. These codes were abandoned during analysis of the first case, as they did not 'fit' the data. The next tactic was to code the data using a framework provided by Crowston, Li, Wei, Eseryel and Howison (2007). However, this coding system was also abandoned because the granularity of their codes (e.g., shared output, prerequisite) is at a higher level than seemed useful. In addition, when using Crowston, et al.'s (2007) codes it proved difficult to maintain the consistency of the level of analysis (organisation level, project level, or individual level).

The next tactic was to define only two broad codes, 'coordination mechanism', and 'dependency' and perform a purely inductive analysis, subdividing these two codes into finer categories when novel dependencies or coordination mechanisms were identified as the analysis progressed.

Two passes were made over each transcript. The first pass was to code all coordination mechanisms used in the project. A second pass was to code all dependencies (actual or potential constraints on action) in the project. Next, a scan was made of all the coordination mechanisms to ensure that each one had an associated dependency. Finally, a check was made to ensure each dependency had an associated coordination mechanism, or no coordination mechanism could be identified. If no coordination mechanism could be identified for a dependency, this was coded as an 'unmanaged

dependency'. In this way, all coordination mechanisms and dependencies and their relationships present in a transcript were identified.

Although initial coding of coordination mechanisms and dependencies was done using the NVivo™ tool, the data were later transferred to a Microsoft Excel™ file. Since each coordination mechanism could address one or more dependences, and each dependency could be addressed by one or more coordination mechanisms, a 2 × 2 table format in Excel™ gave better visualisation of the data, and the relationships between coordination mechanism and dependencies. In this way, primary coordination mechanisms and primary dependencies were identified for each project. These primary coordination mechanisms were then grouped into secondary categories, based on their similar characteristics, thus providing more generic types of coordination mechanism. In like manner, primary dependencies were grouped into secondary categories providing more generic types of dependency. This gave a reduced set of high-level coordination mechanisms that were used to develop the coordination strategy concept for the project.

Project Land was the first case analysed using this process. Once analysis of Land was complete, the dependencies and coordination mechanisms identified in that project provided a fresh list of starter codes for analysis of the next project. The next project was then analysed using the same process as above, and so on for each project in turn.

Identifying dependencies in the project data were not always straightforward. In some instances the participant identified both the coordination mechanism and its dependency in their response by saying *'We do x. If we don't do x, then y will happen'*. A specific example would be, *"We have a daily meeting. If we don't have a daily meeting we will not know the work others in the team have completed, what work they are doing today, and what their current issues are."* Participants seldom explicitly referred to managed dependencies in this way. Malone and Crowston's [2003] advice is to *"identify activities in the process that appear to be coordination activities, and then ask what dependencies those activities manage"* (Crowston & Osborn, 2003, p. 352). Thus, the process for identifying dependencies involved indentifying a candidate coordination mechanism, then isolating the purpose or purposes of the coordination mechanism, then reflecting on the effect on the project if this purpose was not met. This thinking usually led to identification of an underlying dependency that the coordination mechanism was addressing. If no dependency was identified, then the candidate coordination mechanism was dropped. Here is an example illustrating the process:

- From the transcripts, *daily meeting* was identified as a candidate coordination mechanism.
- From the transcripts, activities performed in the *daily meeting* were identified.
- From the transcripts, the purpose of the daily meeting was identified as, *to find out what other people on the team are working on today*.
- The researcher then considered the impact on the project if that purpose was not met; *people would not know who was working on what tasks*. This situation was then categorised as a *task allocation* dependency (defined in Appendix G Glossary of Terms).
- From this information, a dependency: coordination mechanism pair was developed. In this example, the pair would be *task allocation: daily meeting*.

This process for identifying dependencies was used in all instances wherein participants did not directly indicate a dependency when describing a candidate coordination mechanism.

Complexity and Uncertainty

Complexity and uncertainty are concepts in the conceptual framework guiding the study. During analyses, individual instances of complexity and uncertainty influencing actions taken in a project were coded as complexity and uncertainty factors. Their relationship to any coordination mechanisms was also noted.

3.5.2 CROSS-CASE ANALYSIS

Cross-case analysis involves comparing, contrasting, or combining data from all cases to improve the robustness, generalisability, and applicability of findings, and the theory based upon those findings (Yin, 2003b). In this study, cross-case analysis is used to develop a theory of coordination in agile software development projects. The elements of theory are discussed in the final section of this chapter. The theory is explained in steps in Chapter 5 A Theory of Coordination in Agile Projects. First, a coordination strategy concept is developed by combining evidence from each of the agile cases. Second, the influences of factors including project complexity and uncertainty on coordination strategy are explained. Third, a coordination effectiveness concept is developed by amalgamating data from each of the agile cases. In developing this concept, the answers to specific questions asked of each participant at the end of their interview were used. These questions were as follows:

1. What do you think makes *this* project a well-coordinated project?

2. What interferes with the coordination on *this* project?
3. Based on all of your past software project experience, how would you define a 'well-coordinated' project?

Responses are coded as either 'actions' or 'outcomes'. Responses coded as actions contributed to evidence supporting the coordination strategy concept, and responses coded as outcomes contributed to evidence supporting the project coordination effectiveness concept.

Fourthly, evidence for a relationship between the concept of coordination strategy and coordination effectiveness is drawn from case evidence. Finally, propositions are developed based on evidence from the case findings, and the literature. These propositions elaborate the relationships between the coordination strategy and coordination effectiveness, and are stated in such a way as to be testable in future research.

Data from the non-agile case, Rock, is used to show contradictory evidence and further support the applicability of the theory to agile software development projects.

3.6 ETHICAL CONSIDERATIONS

This research involves human participants and their personal and organisational knowledge, therefore Human Ethics Committee approval from the Victoria University of Wellington, School of Information Management Human Ethics Committee was necessary before formal contact was made with any organisations. All research data were confidential to the participant, the researcher, the research supervisors, and the transcribers who signed confidentiality agreements. In addition, data were stored securely to maintain the privacy of individuals and organisations. Following the principle of informed consent, an organisational representative signed a consent form to allow the study of a specific project, and all participants signed individual consent forms. Project leaders gave permission for the researcher to attend meetings and take photographs of the work site. Photographs and documents do not identify participants or contain any identifying details of the project or organisation. Participants were informed in writing and in person that they could withdraw from the study at any time up to six months from the date of the interview. A selected participant from each project reviewed the case study description to check it disguised the organisation and the participants.

3.7 VALIDITY, RELIABILITY, AND RELEVANCE

When a research study meets accepted standards for validity and reliability the research conclusions provide a sound basis upon which future research can build. Qualitative methods literature (Creswell, 2003; Miles & Huberman, 1994; Yin, 2003b) has no definitive rules or guidelines for achieving validity and reliability. Yin (2003b), a specialist in case study research, uses traditional nomenclature on validity and reliability, familiar from statistics-based studies of phenomena. For qualitative positivist case study research, he argues for examining construct validity, internal validity, and external validity, along with reliability.

Construct validity in case study research is problematic (Yin, 2003b). In quantitative research, construct validity is “*the extent to which a given test/instrumentation is an effective measure of a theoretical construct*” (Straub, Boudreau, & Gefen, 2004, p. 424). In this study the ‘constructs’ of interest include dependencies, coordination mechanisms, coordination strategy, and coordination effectiveness. Since this is case study research, instrumentation (e.g. interview protocols) is not defined formally and methods for testing instruments and their effectiveness are not available. Yin (2003b) and Dubé and Paré (2003) make these recommendations for achieving construct validity in positivist case study research: use multiple sources of evidence (also called triangulation), establish a chain of evidence (e.g. use a research protocol), and have key informants review draft case study reports. In this study, triangulation is achieved by involving multiple respondents within a single project, and by using multiple sources of evidence, as explained in the data collection section. A ‘chain of evidence’ is achieved by following a stated protocol and asking participants to review case descriptions. The purpose of these techniques is to show that the theoretical constructs of interest do exist in the situation, and evidence for this comes from multiple sources and is not the possibly erroneous perspective of a single researcher.

Internal validity is concerned with the causal links between constructs. Yin (2003b) notes that only causal and explanatory case studies are concerned with internal validity. This study is an exploratory case study where causal links are not usually a major focus. There is, however, an assumed causal relationship between coordination strategy and coordination effectiveness stated in the conceptual model. To investigate that relationship, an analysis of participant interview data was performed to identify specific statements linking the cause (a coordination mechanism), and the effect (any form of coordination effectiveness).

External validity is concerned with the generalisability of the findings to a larger population. In case study research with four cases, this type of generalisability is not possible. Rather, the findings are presented in a way that allows readers to assess their potential applicability to other settings (Miles & Huberman, 1994). Yin (2003b) recommends replication logic to address external validity. In this study, three of the four cases were selected as literal replications. That is, they were expected to provide similar results, and one case was a theoretical replication, which was expected to provide contrasting results to the other three cases. By creating a detailed description of each case, and elucidating the data analysis process, readers can decide about the applicability (external validity) of the findings to other similar situations.

Reliability is concerned with ensuring a study could be repeated by another independent researcher and the same, or similar, conclusions drawn (Pare, 2004). Yin (2003b) recommends using a case study protocol and a data repository to ensure reliability. This chapter has described the protocols for data collection and data analysis. An electronic project database was maintained throughout the research process and whenever possible paper artefacts were converted to electronic form. All records pertaining to a case were tagged with a unique identifying code and stored in an organised file system for easy searching.

Dubé and Paré (2003) describe 34 practices to achieve rigor and address validity in positivist case study research in information systems. Their set of practices, or attributes, encompass all of the quality criteria mentioned by Yin (2003b) and Miles and Huberman (1994). The following pages contain three tables which show how this study addressed each attribute defined by Dubé and Paré (2003). Only relevant attributes are included (i.e. attributes relevant only to team research and single case research designs are omitted). Table 10 shows how rigor was achieved in design, Table 11 shows how rigor was achieved in data collection, and Table 12 shows how rigor was achieved in analysis. Yin's (2003b) practices for achieving validity are also shown in the tables.

Research relevance is another consideration in a practical field like information systems development (Roseman & Vessey, 2008). This study achieves relevance by selecting on-going or recently completed projects that are typical examples of many such projects in the field, by drawing evidence directly from participants and project work artefacts, and by having research participants review case descriptions to ensure their accuracy.

Table 10 Rigor in the design phase, adapted from Dubé and Paré (2003)

Attribute	Purpose	How used in this study
Clear research questions	<ul style="list-style-type: none"> • Provide a specific focus for the study and indicate where the practical and theoretical contributions are likely • Addresses reliability (Miles & Huberman, 1994) 	<ul style="list-style-type: none"> • There are two research questions each with sub-questions
A priori specification of constructs	<ul style="list-style-type: none"> • Assists in the initial research design by providing initial concepts of interest 	<ul style="list-style-type: none"> • An initial and tentative conceptual model is developed from the literature on software projects and coordination
Clean theoretical slate	<ul style="list-style-type: none"> • Reduces bias in the study analysis • Reduces the chance of the researcher placing limitations on the findings 	<ul style="list-style-type: none"> • There is no <i>a priori</i> detailed elaboration of concepts and relationships. An initial conceptual framework described concepts and relationships in a high-level and general way • Coordination Theory provided guiding principles and was used as a data analysis tool
Multiple case design	<ul style="list-style-type: none"> • Improves the robustness of the developed theory which will be applicable beyond a single case • Case comparison is enabled and case differences enhance theoretical breadth 	<ul style="list-style-type: none"> • Four cases are used in the study
Replication logic in multiple case design	<ul style="list-style-type: none"> • Cases are selected because of their substantive nature or theoretical relevance • Addresses external validity (Yin, 2003b) 	<ul style="list-style-type: none"> • Three literal replication cases of typical agile software development projects are selected • One theoretical replication case - a non-agile software development project is selected
Unit of analysis (UoA)	<ul style="list-style-type: none"> • Relates the case to a broader body of knowledge • Sets a boundary on both the data to collect and the applicability of developed theory 	<ul style="list-style-type: none"> • UoA is the agile software development project with co-located participants • Applicability of the theory is at the project level
Pilot case	<ul style="list-style-type: none"> • A case selected to refine the case protocol and any instruments used 	<ul style="list-style-type: none"> • A pilot case trialled the case study protocol and interview schedule
Context of the case is described	<ul style="list-style-type: none"> • Clarifies the context in which the new theory is applicable • Improves case creditability 	<ul style="list-style-type: none"> • Each case is described in full in a case description using a pre-defined framework

Table 11 Rigor in the data collection phase, adapted from Dubé and Paré (2003)

Attribute	Purpose	How this attribute was met in this study
Elucidation of the data collection process	<ul style="list-style-type: none"> Ensures external parties can understand how data were accumulated, what sources were used and why, and how each source contributed to the findings Addresses reliability (Miles & Huberman, 1994) by ensuring another person could follow the same procedures and arrive at the same conclusions 	<ul style="list-style-type: none"> The methodology chapter fully describes the data sources, data collection procedures, and how each source contributed to the findings
Multiple data collection methods	<ul style="list-style-type: none"> Enables convergence of evidence by providing more than one source of evidence for a phenomenon Addresses construct validity (Yin, 2003b) 	<ul style="list-style-type: none"> Multiple data sources include: individual and group interviews, notes and sketches recorded during interviews, questionnaires, photographs, organisational web site, project artefacts, and observation with field notes
Mix of qualitative and quantitative data	<ul style="list-style-type: none"> Provides a different perspective on the data and may contribute to better theoretical explanations 	<ul style="list-style-type: none"> Majority of data is qualitative. The questionnaire provided some quantitative data on the use of agile practices on the project
Data triangulation	<ul style="list-style-type: none"> Strengthens evidence for findings as they are based on more than one source of evidence Addresses construct validity (Yin, 2003b) 	<ul style="list-style-type: none"> In each case, data comes from multiple sources. Transcripts of interviews of at least two and up to five people directly engaged in each case project are analysed. Whenever possible evidence from more than one source is used to confirm a finding
Case study protocol	<ul style="list-style-type: none"> Ensures external parties can understand what procedures were followed, how data were accumulated, what sources were used and why, and how each source contributed to the findings Contributes to both validity and reliability of the study (Yin, 2003b) 	<ul style="list-style-type: none"> The case protocol includes these documents: <ul style="list-style-type: none"> Participant information sheet Consent forms: organisation, individual Interview schedule for the project leader Interview schedule for the team member A protocol describing the general procedure to follow for each case interaction Questionnaire for project data Questionnaire for agile practices data
Case study database	<ul style="list-style-type: none"> Contributes to the reliability of a study as findings can be traced to original data sources (Yin, 2003b) 	<ul style="list-style-type: none"> Data pertaining to each case is stored electronically and as hard copy (if appropriate) in an easily searchable form

Table 12 Rigor in the data analysis phase, adapted from Dubé and Paré (2003)

Attribute	Purpose	How this attribute was met in this study
Elucidation of the data analysis process	<ul style="list-style-type: none"> • Contributes to an auditable trail of evidence from data to final theory. Improves reliability and reduces bias 	<ul style="list-style-type: none"> • The data analysis process is fully described in the methodology chapter
Field notes	<ul style="list-style-type: none"> • Provides evidence to confirm or extend interview data 	<ul style="list-style-type: none"> • Observations were written down during or shortly after the event. Notes and sketches were made during interviews
Coding and reliability check	<ul style="list-style-type: none"> • Contributes to an auditable trail of evidence from data to final theory. Improves reliability and reduces bias 	<ul style="list-style-type: none"> • Analytical codes were used in the NVivo application to categorise source data into concepts • A reliability check was not performed in this study
Data displays	<ul style="list-style-type: none"> • Contributes to an auditable trail of evidence from data to final theory. Improves reliability 	<ul style="list-style-type: none"> • Tables were used to display quotes contributing to concept definitions
Flexible and opportunistic process	<ul style="list-style-type: none"> • Allows for emerging concepts and relationships to be further investigated either with new cases or with further evidence from a single case 	<ul style="list-style-type: none"> • Project documents were collected, photographs taken, and participant observation was carried out when the opportunity arose (for on-going projects)
Logical chain of evidence	<ul style="list-style-type: none"> • Contributes to an auditable trail of evidence from data to final theory • Improves reliability of information presented • Addresses construct validity (Yin, 2003b) 	<ul style="list-style-type: none"> • Interviews are transcribed and participant ID codes attached to quotes so findings can be traced to their source • All written material (is kept in a research database and ID codes identify the case and the individual participants)
Searching for cross-case patterns	<ul style="list-style-type: none"> • Search for similarities and differences between cases by selecting categories or dimensions for comparison • Addresses internal validity (Yin, 2003b) 	<ul style="list-style-type: none"> • Cross-case analysis is used to compare the findings from each case after indentifying categories in within-case analysis
Use of natural controls	<ul style="list-style-type: none"> • Shows how some factors and not others explain the differences between cases 	<ul style="list-style-type: none"> • Project size (number of project team members) and methodology were controlled for in the selection process
Quotes (evidence)	<ul style="list-style-type: none"> • Provides evidence for the development of theoretical concepts and their relationships 	<ul style="list-style-type: none"> • Quotes are used throughout the case description, within-case, and across-case analysis, to support theory development
Project reviews	<ul style="list-style-type: none"> • Validates factual information and improves reliability • Addresses construct validity (Yin, 2003b) 	<ul style="list-style-type: none"> • For each case, a participant reviewed the project description to verify its accuracy
Comparison with extant literature	<ul style="list-style-type: none"> • Contributes to internal validity • Strengthens the findings of a study because other studies either support the findings, or do not support the findings indicating the theory explains more than previous theories 	<ul style="list-style-type: none"> • Findings from extant research studies are used to support the theory developed in Chapter 6 Discussion

3.8 THE ELEMENTS OF THEORY

The main purpose of information systems research is to develop theories that describe, explain, or predict IT-related phenomena (Gregor, 2006; Weber, 2012). The purpose of this study is to develop a theory about the phenomenon of coordination in the context of co-located agile software development. This theory is presented in Chapter 5 A Theory of Coordination in an Agile Context. Because it is useful to understand what theory is in the field of information systems, this section discusses the elements of theory.

Dubin (1978) initially described five essential elements of theory, which are things (the units, entities, or components of interest), laws of interaction between those things, boundaries, system states, and propositions. Gregor (2006), drawing on Dubin (1978) and the IS reference fields of sociology, psychology, management, and organisation science, defined the elements of theory in IS as:

- A means of representation such as a narrative, model, or diagram
- Constructs, which are the phenomena of interest in a study
- Statements of relationships between constructs
- Scope, which is the degree of generality of the statements of relationships and their boundaries

and optionally,

- Causal explanations
- Testable propositions and
- Prescriptive statements about how to accomplish something in practice

More recently, Weber (2012) added to this discussion by proposing a framework for evaluating theory in the IS field. His elements of theory include:

- Constructs
- Associations between constructs
- States that constructs take – that is, the legitimate range of values of all associated constructs
- Events that constructs undergo which cause a change their state - events are relevant in theory involving dynamic phenomena

Together, these elements describe the boundary of a theory, that is, the phenomena included in the theory.

Weber (2012) also explained that theory has five qualities: importance, level of generality, parsimony, novelty, and falsifiability. The first four qualities can only be assessed subjectively, and by the community to whom the theory is relevant. The fifth quality, falsifiability, depends on the precision with which constructs, associations, states, and events of a theory are defined. Falsifiability can only be achieved if the theoretical predictions are precise enough to be tested empirically (Weber, 2012). In practical terms, this means a theory includes clearly stated predictive propositions suitable for translation into hypotheses (Bacharach, 1989).

When discussing theory, there is a lack of clarity in some of the terminology. Terms in common use include theory, model, theoretical model, and conceptual framework. Dubin (1978) equates theory, model, theoretical model, and system. Whereas, Whetton (1989) considers theory and model to be synonymous. Sutton and Staw (1995) explained this lack of consistency thus:

There is lack of agreement about whether a model and a theory can be distinguished, whether a typology is properly labeled a theory or not, whether the strength of a theory depends on how interesting it is, and whether falsifiability is a prerequisite for the very existence of a theory. (Sutton & Staw, 1995, p. 371)

Shapira (2011), in the field of organisation science, proposes various levels of theoretical development: conceptual frameworks, models, and theory. In his schema, fieldwork produces descriptions of phenomena, from these descriptions initial conceptual frameworks emerge. Conceptual frameworks are the precursors of models, and once tested, a model gains the status of a theory. **Error! Reference source not found.** shows Shapira's (2011) schema.

Table 13 Shapira's (2011) schema of theory types

Conceptual Framework	Model	Theory
1. Provides a structure to organise observations 2. Describes the structure clearly and precisely	1. Is useful, but does not necessarily provide an explanation of observed phenomena 2. Derives predictions based on clearly specified assumptions 3. Is precise and falsifiable	1. Provides a coherent explanation of a set of observed phenomena 2. Derives predictions based on clearly specified assumptions 3. Is precise and falsifiable 4. Is tested by comparing its predictions to reality

Weber (2012) also made a distinction between model and theory but the distinction is not, as Shapira proposes, that a theory is a model that has been empirically tested, but rather, a theory is a model that is precisely specified and is capable of being tested. This

thesis takes Weber's view, that a theory is a precisely defined model of some phenomenon. Because the aim of this thesis is to define a coordination theory as precisely as possible, and in a form suitable for future testing, then it seems appropriate to describe it as a theory.

This chapter develops and presents the theory of coordination by addressing each of the following five elements of theory:

Things, which are the units, entities, or components of interest in a theory (Dubin, 1978; Weber, 2012). Entities are also called concepts and constructs, although IS literature generally assumes a construct is more precisely specified than a concept. This thesis uses the word 'concept' to refer to the entities of the theory of coordination. When a concept is broken down into sub-concepts, these sub-concepts are called components.

Associations, which are the relationships between the constructs in a theory (Weber, 2012). Weber's (2012) *associations* between constructs are equivalent to Dubin's *Laws of interaction* (1978) between concepts. A law of interaction is defined as the connection between the units or things in a theory.

System states, which define how theoretical concepts interact with each other depending on the values they can take (Dubin, 1978; Weber, 2012). According to Dubin "*the system state is defined by the values taken by all the variables or units in the system*" (Dubin, 1978, p. 9), a system state has some persistence over time, and it is possible for a system to have only a single state. Weber (2012) uses a similar idea called a 'state space'. This space sets the boundary of the theoretical system because within this space all possible combinations of values of all theoretical constructs must fall.

Boundaries, which define the limits within which the theory will hold to be true (Dubin, 1978; Weber, 2012). A boundary statement defines applicable context of a theory and to whom, where, and when the theory holds. The boundaries of a theory delineate the extent of its generalisability. Although Whetton (1989) explains that this might not be clear until a theory is tested in different contexts.

Propositions, which are the logical consequences or predicted outcomes of the interactions between concepts in the theory. Propositions can only be specified once the concepts, laws, boundaries, and states of a theory are specified. Dubin calls propositions "*truth statements about the model*" (Dubin, 1978, p. 10). Because multiple theoretical relationships exist between concepts, the number of propositions related to a model can be very large. Therefore, it is important to focus on strategic propositions,

or propositions that point out something notable occurring in the values of one or more concepts. Propositions differ from laws of interaction because, “*the law of interaction tells what the relationship is, and the proposition states what the predicted values of the units will be.*” (Dubin, 1978, p. 170) [emphasis as in the original]. Weber’s (2012) criterion of *falsifiability* is met in stating predictive propositions of a theory.

An additional element of theory is an event. According to Weber (2012), when an event occurs, it causes a change in the state of a system. It is most appropriate to define the events of a theory when the theory is a process theory. Theories in IS are generally categorised as process theories or variance theories (although Burton-Jones, McLean, and Monod (2004) have argued for additional categories including system and hybrid forms). Markus and Robey (1988) defined the difference between process and variance theories as shown in Figure 9**Error! Reference source not found.** Variance theories explain how variation in the values of one construct leads to variations in the values of another, at one point in time. In contrast, process theories explain a sequence of events over time.

The coordination theory to be presented in chapter 5 more closely aligns with the criteria of a variance theory. The theory has two principal concepts and the assumption is when one concept is present, the other will also be present. Further, when the values of the constituent parts of one concept change, then its associated concept will also undergo a change in value. Therefore, defining events in this theory is not appropriate, and they are not considered further.

Creating new theory is called theory building (Eisenhardt, 1989). This process is distinct from that of theory testing, which involves matching an existing theory “*with the real world which it is intended to characterise*” (Dubin, 1978, p. 10). In practical terms, theory testing involves converting propositional statements into testable and falsifiable hypotheses, operationalising constructs, and using those constructs to take measurements in the field. The results of those measurements provide evidence to support or refute the hypotheses. In contrast, theory building is based on observation and description of a context, on existing research literature, on logical conjecture, or on a mixture of these sources. This study builds theory based on field data and existing literature with the aim of presenting a theory in a form suitable for future testing.

	VARIANCE THEORY	PROCESS THEORY
ROLE OF TIME	Static	Longitudinal
DEFINITION	The cause is necessary and sufficient for the outcome	Causation consists of necessary conditions in sequence; chance and random events play a role
ASSUMPTIONS	Outcome will invariably occur when necessary and sufficient conditions are present	Outcomes may not occur (even when conditions are present)
ELEMENTS	Variables	Discrete outcomes
LOGICAL FORM	If X, then Y; if more X, then more Y	If not X, then not Y; cannot be extended to "more X" or "more Y"

Figure 9 Variance and process theories from Markus and Robey (1988)

3.9 SUMMARY

This chapter has described the research paradigm and the research methodology used to address the research questions. The choice of a positivist multi-case study method was justified followed by a description of the overall research process. All aspects of the case study including case selection, data collection, and data analysis were fully described. Ethical procedures and approval were described followed by a discussion of how the study met accepted standards for validity and reliability in case study research. Procedures for ensuring the relevance of the study were discussed. This chapter also contains a section on the elements of theory because the purpose of this study is to develop theory. This theory will be presented in Chapter 5.

The next chapter presents analyses of the four cases making up the study.

4. THE CASES

The purpose of this chapter is to analyse how software projects are coordinated when producing software using a co-located agile development approach. This chapter presents analyses of four software development projects particularly focusing on the dependencies and coordination mechanisms in use, leading to the identification of a coordination strategy unique to each project. Antecedents to the coordination strategy are also identified.

This chapter contains within-case analysis of each case. This chapter has the following structure:

- Introduction
- Case Hour – Pilot case
- Case Land
- Case Storm
- Case Silver
- Case Rock

4.1 OVERVIEW

A pilot case and four full cases form the basis of this research study. The pilot case is code-named Hour and the full cases are code-named Land, Storm, Silver, and Rock. Hour and Rock were not using agile software development. This chapter introduces each case only briefly because full case descriptions are available in appendices. Each case description gives a holistic view of the project and may be read before reading the following case analysis. The case analyses in this chapter, however, stand alone, and make sense without first reading the case descriptions.

The agile projects used one or both of the methodologies, Scrum and Extreme Programming (XP). It is not necessary to have a thorough understanding of these methods to follow the logic presented in this chapter, because practices with a coordinative function are briefly described as part of the case analyses. Useful resources describing the normative agile methods and their practices are provided by Strode (2005) and Williams (2010), and XP and Scrum are summarised in Appendix H Summary of XP and Scrum.

This chapter presents the analysis of each case using a standard format beginning with the project dependencies, followed by the coordination mechanisms, and then the

overall coordination strategy. Table 14 provides an overview of the cases contributing to the analyses. Data sources and the codes used to identify interview quotes are provided in Appendix F Data Sources. All quotes have an identification code in the format [XP01], where X is a unique letter assigned to the project, P is letter code indicating the interviewee's role (P for project leader, T for team member), and 01 is a unique number assigned to the interviewee. To maintain the anonymity of people, organisations, or business units whose names appear in the quoted passages in this thesis, alternative names are used. Substitute names are placed in brackets (e.g. [Sally], [CreditCardCo]).

Table 14 Overview of the four cases

	Cases			
	Land	Storm	Silver	Rock
Organisation type	Government	Commercial service provider	Commercial software development firm	Commercial bank
Organisation size	2000 in NZ	200 in Australasia, Asia and Europe	20 in NZ	5000+
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	To replace and enhance an online system for viewing transaction information
Contractual basis	In-house development	Independent contractors working on the client site	Development for external client	In-house development
Development methodology	Scrum	Scrum and XP	Scrum and XP	Hybrid waterfall/Kanban
Team size	6	10	5	7-15
Interviews	2	5	4	4
Roles of interviewees and their codes	[BP01] Project manager [BT01] Software developer	[CP01] Project manager [CT01], [CT02] 2 Software developers [CT03] Tester [CT04] Domain expert	[DP01a] Development Manager [DP01b] Scrum coach [DT01], [DT02] 2 Developers Note – DPO1a and DP01b were interviewed together	[GP01] Business analyst [GT01] Analyst programmer [GT02] Test analyst [GT03] Technical designer

All terms used in the following case analyses are defined when they first occur, and not thereafter. All of the definitions are summarised in Appendix D Glossary of Terms.

4.2 CASE HOUR: PILOT CASE

Project Hour was a pilot case for the study selected for its convenience. The purpose of this case was to check the viability of the case selection, data collection, and to a small extent, the data analysis process. The project was typical of those selected for the main study although an agile method was not in use. This project involved a major enhancement to an existing software system made necessary by changes in government regulations affecting calculations made within the system. The organisation where the project occurred was not-for-profit and provided a socially important service to the whole of New Zealand. The project involved four people, two managers, and two developers. There was also an extensive user base in branch offices across the country. The project participants followed traditional project management principles during development and did not use any system development methodology. Three people involved in the project were interviewed, the project leader and two software developers. Interviews followed the prepared schedule and some interview data were transcribed although it was not fully analysed since the pilot case does not form part of the thesis findings. After the three interviews were complete the interview schedule was adjusted. The sequence of questions was reorganised to improve the flow of the interview, and redundant questions providing similar responses were removed. These adjustments were very minor.

Before proceeding to analyse the first case, two fundamental definitions from the conceptual framework are repeated here for the readers benefit because they occur repeatedly in the following analyses:

4. A dependency occurs when the progress of one action relies upon the timely output of a previous action, or the presence of some specific thing.
Dependencies lead to potential or actual constraints on projects.
5. A coordination mechanism is an entity (person or artifact) or activity (practice or technique) addressing one or more dependencies in a situation.

4.3 CASE LAND

Project Land occurred in the government department responsible for much of the natural environment of New Zealand. This project was highly important to the organisation, and involved a major enhancement of the organisation's public web site. The aim of the project was to provide a better level of service to the public by creating a dynamic web site, and streamlining of the processing of requests for service. This

involved redeveloping the current static site using Ruby on Rails and a MySQL database, and routing requests to the main database system, and to an external e-payment system.

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, and none of the project team members had experience with agile development. 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.

The team 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. The sprint was organised this way 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. Communication was achieved by 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.

Project issues 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 for the development side of the team while waiting for clarification of requirements.

Stakeholders in the project ranged from the public, the front-line office staff that currently process the applications made by the public, the supervisors and managers of those staff, and the technical support services section that provided technical infrastructure and database support for the system. Because of the importance of the project within the organisation, the stakeholders also included the wider organisational management including the CIO, IT Oversight Committee, and the head of this government department. The team were also stakeholders.

The public was not consulted directly on requirements; instead, the supervisory level staff and their managers acted as proxy end-users for interpreting domain and requirements knowledge for the team. However, to ratify the requirements and provide some feedback from likely end-users, usability testing was carried out with a few members of the public who conformed to the profile of typical users [FN]. Figure 10 is a diagram of all stakeholders, their organisational roles, the primary communication lines, and the direction of the communication links between each role.

Figure 10 Land: stakeholders and their communication links

Data for this case was drawn from a various sources as shown in Table 15.

Table 15 Land data sources

Source	Identifying Code	Detail
Interviews	BP01 BT01	Project manager (a trained systems analyst acting as Scrum Master) Software developer
Web site	WS	Publically available organisation data
Public documents	PD	Publically available annual report
Questionnaire A	QA	Project profile and software development practices
Questionnaire B	QB	Software development practices only
Project documents	CD	Stakeholder breakdown User stories System design
Photographs	PH	None
Sketches	SD	Room layout System architecture sketch
Field notes	FN	Initial meeting with project manager Stand-up meeting User story breakdown session

The full description of this case is in Appendix I Land Case Description.

LAND DEPENDENCIES

There are two forms of dependency in a situation, managed and unmanaged. Managed dependencies have an identifiable coordination mechanism whereas unmanaged dependencies do not. This section first describes managed dependencies in Land, followed by the decision rules for classifying data into dependency categories. Following this is a description of the unmanaged dependencies occurring in the project.

Managed dependencies in Land

Dependencies identified in Land included expertise, requirement, task allocation, activity, business process, and entity (see Table 14 for definitions). These dependencies are called primary dependencies, since they were the initial categories of dependency found in the analysis. To identify these primary dependencies, Crowston and Osborne's (2003) procedure was followed (as described in Chapter 3 Research Methodology). To illustrate how dependencies were identified and categorised an example is provided here. This example is based on the following excerpt from an interview transcript:

“...in terms of who to ask within the business specific questions, yeah, it was in the weekly meetings, I figured out that Brian was doing this, Mary was doing this, and the other Mary had a different role entirely.” [BT01]

This excerpt was coded as a dependency, since a dependency occurs when the progress of one action relies upon the timely output of a previous action, or the presence of some specific thing. In this example, there is a dependency because the developer needs to know who has information relevant to the project in order to carry out his work (i.e. the developer depends upon the presence of a specific thing, which is information in this example). The next step in the analysis was to further categorise the dependency according to its purpose. In this example, the dependency was categorised as an *expertise* dependency because the developer needed to know the role, and therefore the expertise, of the other team members (i.e. ‘the business’). Without this information, the developer would not know who to go to for information about existing system functions or for information on technical support issues. Since the developer became aware of this information by attending weekly sprint planning meetings, then the meetings were categorised as a coordination mechanism. Thus, the sprint planning meeting acted as a coordination mechanism for managing an expertise dependency.

Table 16 provides examples of evidence from project Land data illustrating each primary dependency. The table defines each of these primary dependencies, and explains why the quote is an example of the dependency. The explanations also indicate the coordination mechanism used to manage the identified dependency.

Once primary dependencies were identified using this method, certain commonalities between them were found. This led to new, higher-level, or summary categories of dependency, which were named secondary dependencies. Table 16 also shows the grouping of primary dependencies into secondary dependencies.

Secondary dependencies formed three distinct groups: knowledge, task, and resource dependencies. Requirement, expertise and task allocation were categorised as *knowledge* dependencies because they involve forms of knowledge needed by project participants before they are able to perform project tasks. Activity and business process dependencies were categorised as *task* dependencies as both of these dependencies involve the performance of actions by one group of actors before other project actors could begin their required work. After analysis of all the cases was complete, further categories were found, and they are introduced here for completeness (these dependencies will be defined when they occur in the analyses of cases Storm, Silver, and Rock). These additional categories were technical dependencies and historical dependencies. To complete the dependency categorisation,

entity and technical dependencies were categorised as *resource* dependencies because both involve the necessary availability of people or things, including software components (technical things), which are a form of resource. The historical dependency was categorised as another form of knowledge dependency.

Decision rules for dependency analysis

Classification decisions were necessary when coding data into dependency categories and the following decision rules were applied. Allocation of evidential data to the entity, expertise, or requirement dependency involved specialisation logic; a requirement dependency is a specialised form of expertise dependency, and an expertise dependency is a specialised form of entity dependency. For example, if a person has particular requirements knowledge then this could be coded as, 1) An *entity* dependency because a person is involved, or 2) An *expertise* dependency because the person has particular expertise, or 3) A *requirements* dependency because the person has particular expertise about requirements. In this example, the dependency is coded as a requirements dependency even though the dependency involves both expertise and an entity.

Unmanaged dependencies in Land

Unmanaged dependencies draw attention to missing or inadequate coordination mechanisms in a project. In Land, requirement, activity, and entity dependencies are unmanaged dependencies. Unmanaged requirement dependencies occurred for two reasons. First, information was not readily available because the people with domain or requirement knowledge were not in close proximity to the development team. The project leader in Land explained:

"It would have been so much easier if they were sitting right next to us. But, 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 email 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, yeah."
[BP01]

Table 16 Land dependencies

Secondary Dependency	Primary Dependency	Definition	Evidence	Explanation
Knowledge	Expertise	A situation wherein technical or task information is known only by a particular person and this affects progress	<i>"... in terms of who to ask within the business specific questions, yeah, it was in the weekly meetings, I figured out that Brian was doing this, Mary was doing this, and the other Mary had a different role entirely." [BT01]</i>	The developer discovered the role and <u>expertise</u> (dependency) of the other team members at the <u>weekly sprint planning meetings</u> (coordination mechanism). Without this information, the developer would not know who to go to for information about existing system functions or for information on technical support issues.
	Requirement	A situation wherein domain knowledge or a requirement is not known and must be located or identified and this affects progress	<i>"So, in our sprint planning meetings we would figure out what was needed for the next sprint and do some kind of design work on what the screen might look like, exactly what validation was required, those kinds of things." [BT01]</i>	The whole team sat together in the <u>iteration planning meeting</u> (coordination mechanism), planning the tasks to complete in the coming iteration and working together to formalise requirement details. Without this session, the developer and other team members would not know the <u>requirement</u> details (dependency).
	Task allocation	A situation wherein who is doing what, and when, is not known and this affects progress	<i>"Each day, so I guess I would coordinate with Sam, usually first thing in the morning we would kind of get together and decide, and have quick catch up about what I was working on, it could have been something I was working on from the previous day." [BT01]</i>	The 'technical' team, with desks side by side, would have a <u>talk each morning</u> (the coordination mechanism) about the current work. Without this session, the current work underway and <u>who is working on particular tasks</u> (dependency) would not be known to the other members of the technical team.
Task	Activity	A situation wherein an activity cannot proceed until another activity is complete and this affects project progress	<i>"Sam waited. He would do the first line of testing. So he would always do an initial user acceptance testing to verify that everything was OK. Graeme would wait for me to complete the work so he could apply his styling." [BT01]</i>	One team <u>member transferred a completed software component</u> (dependency) to another team member with different expertise to carry out additional tasks involving the component, such as user acceptance testing or interface design. There is no coordination mechanism at the project level here. Waits were indeterminate.

Secondary Dependency	Primary Dependency	Definition	Evidence	Explanation
	Business process	A situation wherein an existing business process causes activities to be carried out in a certain order and this affects progress	<i>"...are translated onto what you did online. The flow the user was taking through. So there was a certain dependency inherent in the process, so basically you were choosing something to apply for, then you had to provide some details, then you had to pay. If you break it down like that...that kind of naturally ordered the [development] tasks into, having to pick something first, so we could do things like work out the price, and the other bits and pieces that need to be filled in.. " [BT01]</i>	Development tasks were organised around the existing business process used to handle applications. This was because the output of one part of the process acted as input to the next part. This was managed by assigning each portion of functionality to an <u>iteration</u> (the coordination mechanism) in the <u>same order that it occurred in the actual business process</u> (dependency).
Resource	Entity	A situation wherein a resource (person, place or thing) is not available and this affects progress	<i>"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 she had built." [BP01]</i> <i>"Project System Support, essentially system administration yeah. I would have to occasionally wait on them to do something for me. Like I needed something installed on a server, or I needed access to a server or something like that." [BT01]</i>	The <u>developer is not available</u> (dependency) affecting <u>iteration activities</u> (coordination mechanism) <u>A server is not available</u> (dependency) causing the developer to wait. There is no coordination mechanism here because the wait is indeterminate.

The second reason concerns availability. Team members were not available at all times. There was only a single person whose work was devoted to the project full time; although he did not have a full-time role. All other team members were working on multiple tasks and other projects simultaneously. This made gathering and verifying requirements problematic.

"Sometimes I would have to wait for the business on the design...not the visual design but, you know...on this particular page, this field is required, this field is optional, that kind of stuff." [BT01]

"Now in terms of our daily team meetings, 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, and also the lack of, and also the competing priorities that they had. Which was another major issue for the project that they all had other really equally important, if not more important, stuff going on, so trying to get them to be available at the start of every day to review what we did yesterday and what were going to do today was not really that possible." [BP01]

"Because, so much of these kinds of these things [team members having multiple priorities] in [the organisation] involve people being in meetings, and so trying to get that regular face-to-face communication when [Brian] is in an all day meeting 4 out of 5 days in a week, made it really hard". [BP01]

The unmanaged activity dependency involved team members waiting for work to be completed before they could perform their allocated work, for example:

"[Steve, the business analyst] waited. He would do the first line of testing. So, he would always do initial user acceptance testing to verify that everything was OK. [Graeme, the web designer] would wait for me to complete the work so he could apply his styling." [BT01]

Two entity dependencies were unmanaged and retarded project progress. The first was interaction with a legal team who checked aspects of the work, and the other was interaction with the IT support section.

"I had to wait sometimes on the business to finalise the copy...So within the application there is text everywhere. And they would tell me what that copy was. So towards the end there was a lot of waiting for final bits of text to be confirmed, from a legal perspective...Grammatically." [BT01]

"I would have to occasionally wait on them to do something for me. Like I needed something installed on a server, or I needed access to a server." [BT01]

Another unmanaged entity dependency involved outsourced tasks. Although these tasks were managed by contracts such as Request for Change applications, they are categorised as unmanaged dependencies because the project team had to wait an indeterminate length of time for a requested resource:

"[X] was a vendor...and occasionally we were waiting on them... So, towards the end, when we were getting towards deployment, they were responsible for provisioning a server to deploy the code onto. So yeah, we just had to wait occasionally. We would send them an email. We would produce a RFC, request for change. Send them an RF, and then they would have to provision the server. And then there [were] a couple of times when we wanted change to the server. I mean it wasn't very long, but there were waits occasionally"
[BT01]

It was possible using this analytical coding process to categorise a dependency as both managed and unmanaged. This happens because there are many instances of a dependency occurring over the life of a project. Some of these instances are managed with a coordination mechanism, but in other cases, they are unmanaged. As this discussion of managed and unmanaged dependencies has shown, this occurred with the *requirement* dependency. At some times during the project, specific knowledge of a requirement was managed with a particular coordination mechanism. For example, a 'user story prioritisation session' is a coordination mechanism for identifying the relative importance of a requirement. At other times during the project, knowledge about the priority of a requirement was not known, and considerable waiting was involved before the person who could explain the requirement was available to make a decision on the requirement's priority. In this instance, a requirement dependency occurred and there was no identifiable coordination mechanism for managing it.

Summary of dependencies in Land

Land had primary dependencies of the following type: requirement, expertise, task allocation, activity, business process, and entity. Table 17 summarises the primary and secondary dependencies present in Land.

Table 17 Land summary of dependencies

Secondary dependencies	Knowledge	Task	Resource
Primary dependencies	Requirement Expertise Task allocation	Activity Business process	Entity

LAND COORDINATION MECHANISMS

Land used a variety of coordination mechanisms to address project dependencies. An activity or artefact was categorised as a coordination mechanism only when it was addressing one or more distinct dependencies. After identifying primary coordination mechanisms in the data (shown in Table 18 and Table 19), these coordination mechanisms were grouped into secondary, or summary, categories based on their

similarities. These secondary categories of coordination mechanism were identified as synchronisation activities, synchronisation artefacts, boundary spanning artefacts, proximity, availability, and substitutability. Each of these categories is now described.

Synchronisation activities

Synchronisation activities are team-wide activities that bring all, or most, project team members together at the same time and place for some pre-arranged purpose.

Synchronisation activities occurred at different frequencies: per project, per iteration, daily, and ad hoc. A project frequency activity occurs once during a project, an iteration frequency activity occurs once in each iteration, a daily activity occurs once each day, and an ad hoc activity occurs as and when necessary. Table 18 shows all primary coordination mechanisms categorised as synchronisation activities in Land.

Table 18 Land synchronisation activities

Primary Coordination mechanism	Frequency	How this coordination mechanism is used in Land
Iteration zero planning session	Per project	Formal session with the whole team to write user stories and discuss technical and other aspects of the new system
Weekly iteration	Per iteration	Regular, time-boxed set of activities occurring within the weekly sprint
Weekly iteration planning session	Per iteration	Regular informal planning session with the whole team present
Progress tracking with user stories	Per iteration	An activity when the whole team discusses the current sprints' user stories and their current completion status (waiting to be completed, ready for testing, or done).
Story point prioritising	Per iteration	An activity when the whole team discusses the user stories, their scope, relative priority, tasks allocated to the story, task duration (size)
Daily team session	Daily	Daily, informal meeting with the whole team present
Software release	Ad hoc	A functioning version of the application under development was made available to the group of end-users or proxy end-users

Synchronisation was initially achieved with an iteration zero for project planning and generating initial high-level scope requirements. Thereafter, weekly iterations maintained project synchronisation with both a weekly and daily cycle of well-structured activities. For example, a key activity was the weekly iteration planning session, which the whole team was encouraged to attend, and where a variety of activities were carried out. This session was scheduled at the same day and time each week and activities included reviewing progress, viewing software prototypes, providing feedback on the state of the software, sharing domain expertise and

requirements knowledge, and defining, refining, sizing, and prioritising user stories (high-level requirements). The meetings were described as “*a bit of a show and tell and final debrief*” [BP01] and they ran for “*for as long as it would take*” [BP01].

The project manager was pleased with the affect of these sessions, commenting:

“...and that whole process actually worked very well, I was very pleased with it. It built a natural rhythm, and I found I had to nag less as time went on, because people knew that. We would discuss stuff that was needed on the Thursday that Jack would need on the Monday to get going. And ... I am used to then having to remind them several times they are supposed to be getting that, but that over time started becoming a habit. Because they knew that, well for a start, they knew Thursday morning that they would have to be ready with any of their comments that they wanted to input. And any of their thoughts about what they were going to do next. They knew that Friday they would have to be testing. And they knew that the next iteration would start Monday... So, basically, because people work well to deadlines really. People need deadlines really, probably it's not so much 'like', they need them. We, in essence, had a weekly deadline”. [BP01]

These weekly sessions also helped build team responsibility and focus,

“...it did mean that the coordination side of things was a little less intense, than sometimes I have had to do in the past in terms of nagging, because it did build a natural rhythm... [Sarah's] boss, his feedback was that [Sarah] was very concerned about letting the team down. So there was also that sense of 'I am one of the team, and if I don't get the stuff done in time then Jack can't do what he needs to do and that is going to let the team down, and break the rhythm of iteration because you'll get behind and we won't be able to catch up'. Because when you have only got a one week iteration there is not a lot of room in there.” [BP01]

Synchronisation artefacts

Synchronisation artefacts are physical things generated during synchronisation activities. They store project information such as design decisions and may serve as objects to focus discussion. The nature of an artefact can be publicly visible to the whole team at a glance (e.g. on a whiteboard in the team workroom), or largely invisible but available (located in a shared computer file). An artefact can be physical or virtual, temporary or permanent.

In Land, artefacts included the working software, user stories, a product backlog, a design specification document, and a burn-down chart. These artefacts were permanently and publicly available in a shared file repository. Table 19 shows how synchronisation artefacts were used in Land.

Table 19 Land synchronisation artefacts

Primary Coordination mechanism	How this coordination mechanism is used in Land
Task repository/ Product backlog	A list of epics and user stories waiting to be addressed
Design specification document	A document created from decisions made during weekly iteration planning sessions and primarily used by the developer. The developer considered this to be an important resource, which he described as the, “one source of truth” for design decisions [BT01].
Working software	A semi-complete executable version of the system under development provided to the end-users for testing
User stories	A single high-level requirement. Traditionally user stories are written on cardboard cards or cardboard index cards and often follow a particular format. User stories and their priority were created in the weekly iteration planning session, and recorded in a design specification document.
Burn down chart	A chart used to show the progress of story completion over time

Boundary spanning artefacts

Boundary spanning artefacts are artefacts produced to enable reciprocal interaction between project team members and other parties external to the project in order to meet project goals. Land used a single artefact for this purpose. A Request for Change form was produced when a technical change was needed to the existing system to accommodate implementation or testing of the new system. This form was sent to an external vendor for processing. This coordination mechanism was inadequate because although a request was sent, the slow response caused a schedule delay when the request is not fulfilled quickly enough to fit with the project schedule.

Proximity

Proximity is the physical closeness of individual team members. A team can include development team members, customer team members, or other stakeholders. Proximity can range from adjacent desks, to members located in different rooms, floors of a building, different buildings, different countries, or time zones. In Land, proximity by co-location was not achieved for the whole team. The team was in the same building but only the ‘technical side’, had adjacent desks. Team members belonging to the ‘business side’, were on different floors within a large multi-story building. Therefore, the team was not strictly co-located. This caused problems in coordinating development tasks because coordinating with non-co-located team members was time-consuming and there was lag-time in responses to requests for project information, such as requirements.

Availability

Availability is achieved when team members are continually present and able to respond to requests for assistance, information, and to participate in project activities when needed. The organisation where Land occurred used a matrix (Galbraith, 1974) or multi-project organisational structure whereby individual team members work on a number of projects simultaneously. At the project level, these caused a problem because some members of the team were not all working full-time on the project but were working simultaneously on a number of different projects; therefore, they were not always available to discuss project requirements and issues.

Substitutability

Substitutability is achieved when a team member has the expertise and skills to perform the tasks of another to maintain the project schedule. Substitutability is achieved with a coordination mechanism named '*redundant skill*', which enables one project member to take over some of the work of another to maintain time schedules. This happened when, for example, the project manager was able to do database work for the developer when time was short and she had the background knowledge to do this work: "...essentially [Sally] took some of that stuff on, of making sure the data were in the right format in the right place." [BT01].

LAND COORDINATION STRATEGY

Each software project has a combination of coordination mechanisms to address its dependencies. The combination in Land is shown in Table 20. Also shown in this table are primary and secondary dependencies, primary and secondary coordination mechanisms, and managed and unmanaged dependencies. Table columns show dependencies and the rows show coordination mechanisms. Each coloured cell indicates a dependency: coordination mechanism pair identified in the case data. Each pair represents one or more instances of evidence. That means each pair is supported with one or more pertinent sections of interview or other data, for example one or more quotes. The call-out boxes show examples of quotes coded to different cells.

Table 20 Land dependencies and coordination mechanisms

			Dependencies				
			Resource	Task		Knowledge	
			Entity	Business process	Activity	Requirements	Expertise
						Task allocation	
Coordination Mechanisms	Synchronisation activities	Iteration zero planning session					
		Weekly iteration	●	●			
		Iteration planning session				●	
		Progress tracking with user stories					
		Story point prioritising					
		Daily team session					
		Software release					
	Synchronisation artefacts	Product backlog			●		
		Design specification document					
		Working software					
		User stories					
		Burn-down chart					
	Boundary spanning artefact	Request for Change	●				
	Availability	Single priority team					
	Proximity	Co-locate team					
	Substitutability	Redundant skill					
Unmanaged dependency			●	●	●	●	●

"We did a one-week iteration. The reason for the one week iteration, is our developer, as I said, was part time working only 4 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." [BP01]

"... as part of our Thursday morning session we would talk through and actually look at the screens of what had been built that week...But maybe, ..., if we had some question marks. Like what we thought we were going to do, when the developer was starting to work on it and didn't like it or had some questions, we would just get people to look at whatever state it was currently in, and say 'you sure about this?' [and they would say] 'we think that actually, now we see it on the screen, that this is not a great idea'. [BP01]

There are two potential views of the information provided in Table 20, a dependency-based view, and a coordination mechanism-based view. Each view provides different information. Note that the numeric values presented in analyses (in this case and all others) are indicative only and not precise, because they are based on a qualitative assessment.

A dependency-based view provides the cell count data shown in Table 21 and Table 22.

Table 21 Land dependency percentages – secondary dependencies

Secondary dependency	Count	Percentage (n=22)
Knowledge	18	82%
Task	2	9%
Resource	2	9%
TOTAL CM: dependency pairs	22	100%

Table 22 Land dependency percentages - primary dependencies

Primary dependency	Count	Percentage (n=22)
Requirement	11	50%
Task allocation	5	23%
Expertise	2	9%
Activity	1	4.5%
Business process	1	4.5%
Entity	2	9%
TOTAL CM: dependency pairs	22	100%

This view indicates that 82% of dependency: coordination mechanism pairs in Land were associated with a knowledge dependency. That is, attending to requirement dependencies, expertise dependencies, and task allocation dependencies constituted most of the coordination work in Land. Of those knowledge pairs, 61% are associated with requirements. Requirement dependencies involved 50% of all pairs in the analysis. Task allocation dependencies were the next most common type of dependency pair, at 23%.

A coordination mechanism-based view provided the cell count data shown in Table 23.

Table 23 Land coordination mechanisms

Secondary coordination mechanism	Count	Percentage (n=22)
Synchronisation activities	11	50%
Synchronisation artefacts	7	32%
Boundary spanning artefacts	1	4.5%
Availability	1	4.5%
Proximity	1	4.5%
Experience	1	4.5%
TOTAL CM: dependency pairs	22	100%

This view indicates that synchronisation activities, at 50% of pairs, are the predominant coordination mechanisms in Land. When synchronisation activities and their associated artefacts are considered together, they account for 82% of pairs in the project.

In Chapter 2, in the section describing the research questions, an assumption was made that a coordination mechanism that addresses multiple dependencies is more efficient than one addressing a single dependency. Table 20 shows that, although there are 16 individual coordination mechanisms in Land, 13 (about 81%) of them address only a single dependency. Coordination mechanisms addressing multiple dependencies simultaneously are the weekly iteration, iteration planning sessions, and the product backlog.

In summary, Land used a strategy, as depicted in Figure 11, made up of the following coordination mechanisms:

- Synchronisation activities at all levels, project, iteration, daily, and ad hoc to achieve a common understanding of the requirements, and who was doing what, among project team members
- Synchronisation artefacts produced or used during those activities. This is shown in Figure 11 with a double headed arrow between synchronisation activities and artefacts. This is to indicate that synchronisation activities produce artefacts, and those artefacts are used as the project progresses (e.g. stories generated in a meeting are placed in a shared document for later development).
- Boundary spanning artefacts for coordinating with external parties
- Availability of team members. This was a problem in Land caused by their matrix organisation structure. In Figure 11, this is shown with an arrow

pointing from the box labeled organisation structure to availability. This indicates that organisation structure influenced availability.

- Proximity of project team members, another problem in Land caused by the matrix organisation structure. In Figure 11, this is shown with an arrow pointing from organisation structure to proximity. This indicates that organisation structure influenced proximity.
- Substitutability whereby multi-skilled project team members could perform each other's tasks to maintain the schedule

Figure 11 Land coordination strategy

The coordination strategy in Land has two predominant characteristics. Firstly, knowledge dependencies are the main dependencies in the project and those knowledge dependencies are primarily about requirements. Secondly, synchronisation activities make up most of the coordination mechanisms employed to manage those

dependencies. Therefore, the coordination strategy in project Land is dominated by the performance of activities to synchronise knowledge within the team about requirements.

4.4 CASE STORM

Project Storm took place in a quasi-public sector organisation (an SOE¹⁰). 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. After extensive consultation throughout the organisation, he divided the project into seven inter-related sub-projects; each with a different group of end-users. He also selected a team of 10 contractors to develop the new system. Team members were selected because they possessed strong technical ability in Java, 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 (at the time of data collection) 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. But they worked on shifts, so it was not always possible to consult with a particular individual.

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, later adjusted to a one-week timeframe. Typical Scrum practices were used, including 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 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

¹⁰ 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 Apr 2011. <http://www.treasury.govt.nz/glossary/soe>

iteration, and daily, were interspersed with ad hoc meetings between team members, and also between team members and the engineers.

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. 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, or by sending out a team member to work in other business units for a period. Occasionally, hold-ups occurred when there was critical work occurring in the engineers group, as that work always took priority over consultation with the development team. This project was viewed by all of the interviewees as highly challenging due to the complexity of the existing system, and the unique and complex problem domain.

Project storm was complex and involved a variety of stakeholders that included the following groups. The Storm team were the people carrying out the development. When the project began, they were all contract staff. At the time of data collection, two permanent staff had recently joined the team. These staff came from the Central team and the Application team, and their role was to act as domain experts because of their technical and requirements knowledge, and to learn about the new system.

Another group were the engineers. Engineers were the end-users of one of the sub-systems. There were about 60 to 80 engineers located in a room next to the Storm team. Each engineer performed the same role. These end-users worked on shifts because their work outputs were necessary 24 × 7. Beta testers were four engineers who were consulted most frequently. They were preferred people to approach for requirements clarification and testing. The tester instigated this arrangement with the engineers to improve communication and the speed and consistency of response from the engineers. Another group of end users were the Racers who were end-users of the Racetrack application.

Switch vendor was an external organisation providing a product that converted output from the Startpoint/Endpoint sub-system to an international format. The switch product needed to be enhanced by the switch vendor to comply with the needs of the development team. This vendor was located in another country some time zones away.

There was a steering committee. This was a group of permanent management staff to whom the project manager reported at monthly intervals. Another group was the

International communication standards setting body that provided standards for communication. The Storm project applications needed to comply with those standards. Central team were a group of permanent employees who worked on maintaining and enhancing the legacy system. The application team were a group of permanent employees who worked on maintaining and enhancing applications interacting with the legacy system. There was also an external software vendor providing a software product used by the team for data conversion. The vendor was located in South Africa and the product sometimes needed enhancements or bug fixes which meant the team had to communicate with this vendor and wait for updated versions of the software to be available. A diagram of the stakeholders and the main communication links between them is shown in Figure 12.

Figure 12 Storm: stakeholders and their communication links

Table 24 shows the sources of case data.

Table 24 Storm data sources

Source	Identifying Code	Detail
Interviews	CP01 CT01 CT02 CT03 CT04	Manager/project manager/Agile mentor and coach Senior analyst programmer Java Delphi developer Senior software developer Test specialist
Web site	WS	Publically available organisation data
Public documents	PD	Publically available annual report
Questionnaire A	QA	Project profile and software development practices – not completed
Questionnaire B	QB	Software development practices only
Project documents	CD	None
Photographs	PH	Wallboard (2) Done list
Sketches	SD	Room layout
Field notes	FN	Initial meeting with project manager Stand-up meeting User story breakdown session

A full description of this case is in Appendix F Storm Case Description.

STORM DEPENDENCIES

Managed dependencies in Storm

Overall, Storm had a total of seven managed dependencies. Of these, four were *knowledge* dependencies: requirement, expertise, task allocation, and historical. The dependency named 'historical', did not occur in Land, but is apparent in Storm. A historical dependency occurs when knowledge about past decisions is needed and if it is not readily available, this affects project progress. In Storm, this dependency was due to a lack of documentation about the legacy system, and the need to go to resident experts in the organisation who understood why and how the legacy system processed data, and whether certain functions offered in the legacy system needed to be migrated to the new system. A historical dependency is a type of *knowledge* dependency because it involves a form of knowledge needed by project participants before they are able to perform project tasks.

Storm had a single *task* dependency, which is an activity dependency. *Resource* dependencies include entity and technical dependencies. Table 25 provides illustrative evidence for each primary dependency.

Unmanaged dependencies in Storm

Storm had five unmanaged dependencies of the following types: expertise, requirement, activity, entity, and technical. Unmanaged *expertise* dependencies occurred when decisions had to escalate to management and the project team had to wait for the outcome:

"I guess the only one I did not cover was [Atlas - another system]... We are kind of waiting on work to be done on that. But [it] is not so much that the guys are actually working on that. We are waiting on a decision to be made about whether they should work on it." [CT01]

"So who is making that decision?" [Researcher]

"[David] and other management types." [CT01]

Unmanaged *requirement* dependencies occurred when unexpected major requirements arose:

"And every now and then a new one pops up because ... a person will just say 'oh you guys are going to be migrating that; did you know about this package that also does this'. So it is scarily organic at times and that's just an artefact of the [fact that the] whole system is not really documented so you can't just take a snapshot of something and go figure it out..." [CT04]

Unmanaged *activity* dependencies occurred when activities could not proceed without feedback from the end-users (the engineers) and the only mechanism for getting the engineers to test the software was persistence and nagging:

"And as we get more serious [and]...ready to deploy into production, we entice the whole team, the rest of the [engineers] to start using their non-migrated tool and the migration tool in parallel...And that way they will actually use it in anger¹¹ and they ... oh we ... flush out quite a few last few bugs there where it's 'I tried to enter this data that I would have normally have sent and your tool wouldn't allow it'. Whereas you can't really expect someone to figure that out in the testing..." [CT04]

"So how do you...entice them..." [researcher].

"We recommend that they do it, and maybe three more will start doing it, and will report back, and then we say 'it's about to go live in two weeks' time', and we warn them, you know 'if you haven't experienced it already have a look at it and by the way this is going to be the final product so you better like it and get your feedback in now'.... it's quite nagging...And we also get their boss to, he sends out another reminder I guess and he's quite eager

¹¹ In the software development community, when a piece of software or technology has been used in its real target environment they say it has been used "in anger". See <http://www.inanger.com/about/>

for them to use it, and I've heard him saying more than once. 'You know that if you don't tell them now, you're gonna hate it, you know, so get involved early'...And I guess he's already experienced it with everything else that's been developed for them. He probably gets quite tired of the end users saying 'it wouldn't work', saying, 'well why didn't you tell them four months ago'. [CT04]

Unmanaged *entity* dependencies occurred when resources, such as servers, were acquired using a request-and-wait method:

"...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]. [CP01]

Table 25 Storm dependencies

Secondary Dependency	Primary Dependency	Evidence	Explanation
Knowledge	Expertise	<i>"... we can just yell out over our shoulder and grab someone and also if they come to a complex design decision then they should bring all of the team to the table so that everyone hears the design decision that is made." [CP01]</i>	The <u>team members learn what the design decisions are</u> (dependency) at an <u>ad hoc team session around the table</u> (coordination mechanism)
	Requirement	<i>"We have a person on our team, who is one of the gurus, and he sits with us, and we annoy him constantly to get stuff like that. [CT01]</i>	There is a domain <u>specialist located with the team</u> in the same room (coordination mechanism) who can be readily <u>consulted on requirements</u> (dependency)
	Task allocation	<i>"Well you learn that from the daily interaction with them and like the "stand ups" and what they've done there the previous day or whatever and you soon learn that these guys do that, and those guys do that." [CT02]</i>	The team learns <u>who is doing what and when</u> (dependency) at the <u>daily stand up session</u> (coordination mechanism)
	Historical: A situation wherein knowledge about past decisions is needed and this affects project progress	<i>"And then a whole story is done according to that list [the Done list]. Even though that list is still in flux. That is something we only came up with recently because we were finding some things were being, how best to describe it, you would think some stuff was done, but it hadn't been...No, it was more the integration amongst the systems, that were getting on my nerves, at least. Just people thinking that they had done something, and they had done it in one part of the, and you know they had written all the stuff in one part of the system, but they had not actually done the integration with other things to see that it was actually really working." [CT01]</i>	The <u>done list</u> (coordination mechanism) is used to ensure that <u>knowledge about past work</u> (dependency) is signalled to the team.
Task	Activity	<i>"It's true actually, and I think that that's what we are getting much better at doing in our weekly breakdowns or bi-weekly breakdowns is prioritising things so there is minimal blockage happening for everyone." [CT04]</i>	The <u>work of one team member is not blocked by the work of another</u> (dependency) because the <u>stories are prioritised</u> (coordination mechanism) in an appropriate order
Resource	Entity	<i>"...the main people we annoy with that are the operations group, are the infrastructure, the people who look after</i>	The <u>resource, e.g. a server, is not available as and when needed</u> (dependency). There is no coordination

Secondary Dependency	Primary Dependency	Evidence	Explanation
		<i>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]. [CP01]</i>	mechanism in this situation the project team must make a request and wait for a response
	Technical: A situation wherein a technical aspect of development affects progress, such as when one software component must interact with another software component and its presence or absence affects project progress	<i>"...we should be able to include a whole bunch of our modules and the [Vole] modules and just make one application that you deploy onto the desktop, and doing that, helping ourselves by not having to deploy two applications, more than actually getting through work that gives the user [value], give us more things migrated....There are story cards for it, but we keep trying to avoid them, if there was just jokes about that." [CT01]</i>	User stories, in the form of story cards (coordination mechanism), are used to insert work tasks into the project that involve work for another external team because there is a technical dependency between the work of the two teams. <u>Their work must 'fit' together at a later date</u> (dependency)

Unmanaged *technical* dependencies involving external organisations were also managed with a request-and-wait method:

"Yes, but [SOS] is a product that we have brought from South Africa...There is a rep from those guys who looks after us. And we do find bugs and stuff, so we are in contact with him asking for things to be fixed when we find bugs. They are pretty quick at doing it but it means as soon as you find something that you can't fix in there; it is going to be three days before you can think of getting a change." [CT01]

Each of these dependencies involved interactions with business units or other organisations external to the project team. There was no distinct mechanism for coordinating with those units beyond making a request, and waiting for a response. Request-and-wait is not categorised as a coordination mechanism because, in this thesis, a coordination mechanism as an entity (person or artifact) or activity (practice or technique) addressing one or more dependencies in a situation. And, in this situation, no person or action within the project boundary was actively involved in managing or addressing this dependency with an external party. Therefore when a dependency is addressed with a request-and-wait method, this is categorised as an unmanaged dependency.

Summary of dependencies in Storm

Table 26 summarises the dependencies identified in Storm.

Table 26 Storm summary of dependencies

Secondary dependencies	Knowledge	Task	Resource
Primary dependencies	Requirements Expertise Task allocation Historical	Activity	Technical Entity

STORM COORDINATION MECHANISMS

Storm used various coordination mechanisms to address project dependencies. Numerous primary coordination mechanisms were identifiable, and were grouped into the following secondary, or higher-level, categories: synchronisation activities, synchronisation artefacts, boundary spanning activities, boundary spanning artefacts, availability, proximity, substitutability, and coordinator role. Each of these categories is now described along with sample evidence.

Synchronisation activities

Synchronisation activities are shown in Table 27. Synchronisation activities occurred per project, per iteration, daily, and ad hoc.

Storm was divided into sub-projects. In the first sub-project there was an iteration zero to determine epics, stories, and story prioritisation. During data collection, another sub-project was underway involving a different group of end-users (the engineers). This new sub-project did not have an iteration zero because the engineers were not available for consultation as a group because they all worked on shifts. Therefore, story development in this sub-project was more flexible; stories emerged as the project progressed and there was an emphasis on continuous consultation with the engineers to elaborate epics (by decomposing them into stories) and story details. This also explains the two levels of breakdown session, per iteration and ad hoc, as story breakdown sessions occurred both regularly at the start of each iteration, but also whenever there was a shortage of stories displayed on the wallboard for developers to work on.

Table 27 Storm synchronisation activities

Primary Coordination mechanism	Frequency	How this coordination mechanism was used in Storm
Domain specialist on team	Per project	A domain specialist was added to the team to provide additional requirements knowledge about the legacy system. This person is located in the project room and continuously available for consultation by everyone on the team.
Assignment of story to self	Per iteration	At the start of an iteration, team members select stories to work on
Breakdown session (iteration)	Per iteration	User stories for the iteration are selected from the backlog and decomposed into tasks that are 'sized' by the team. Then each story and its tasks are placed on the wallboard in order of priority
Daily stand up	Daily	A session when the whole team stand up together in the project room and take turns to say what they are currently working on, what they did yesterday, and any problems they currently have
Cross-team talk	Ad hoc	Project team members can ask questions of everyone else in the project room, and all the team members can overhear the question and the response and any other conversation about the project
Pair program session	Ad hoc	When a programmer has a problem with the code they can ask for help and another more experienced programmer sits with them to help them with the problem
Breakdown session (ad hoc)	Ad hoc	At any time during an iteration, when few stories remain on the wallboard, additional stories are drawn from the backlog. This means calling an ad hoc breakdown session with the whole team, or a subset of the team. This could also involve consulting with one or more engineers to clarify story details before decomposing the story into tasks
Informal face-to-face negotiation	Ad hoc	Discussion, negotiation, and decision making involving all, or most of, the team members
Continuous build	Ad hoc	Continuous integration involves continual 'builds' (integration and execution) of all system components. If the build 'fails' then there is a bug or other problem with the code base that must be resolved

The complexity of the project caused the team to change the frequency of synchronisation activities as the project progressed. The team found that they were spending too long (many hours) on task breakdown sessions and this was tedious and time consuming. They resolved this problem by changing from a two-week sprint to a one-week sprint. This increased the number of iterations in the project overall, but decreased the duration of the breakdown sessions, and the scope of the work within an iteration. That is, the number of stories broken down in a breakdown session was reduced which decreased the duration of each story breakdown session. This also reduced the number of stories and tasks in an iteration.

Synchronisation artefacts

In Storm, synchronisation artefacts were found to include the product backlog, the wallboard, user stories and tasks, a burn down chart, the Done list, a whiteboard for temporary design work, the working software, and the project wiki. Except for the product backlog, the working software, and the wiki, all of the project artefacts were continuously visible because they were on the wallboard easily seen from all parts of the project room. The whiteboard was for temporary display of information, but if that information was important enough, it was transferred to the wiki for storage:

"Most of the time ... that whiteboard will stay up for a few days before the next design discussion happens or the next whiteboard discussion happens, and then in that case people can look over their shoulder and see what they drew, and get on with actually coding to that plan. And if they think it is intrinsic to the way that the system works then we maintain diagrams and loose informal documentation on a wiki, which is our kind of final place of documentation. And the wiki has a diagramming tool in it that you update your diagrams to whatever the design was now..." [CP01]

"Normally just whiteboard it up and if it is significant enough it will make its way into the wiki." [CT01]

Synchronisation artefacts used in Storm are shown in Table 28 along with an explanation of how they were used in the project.

Table 28 Storm synchronisation artefacts

Primary Coordination mechanism	How this coordination mechanism was used in Storm
Product backlog	A list of epics and user stories waiting to be addressed. This is stored in an Excel spreadsheet and made available to the whole team on the wiki
Wallboard	A whiteboard, clearly visible to all of the team, on the wall in the project room displaying current work in the form of stories, tasks, the progress of tasks, the allocation of tasks, the burn down chart, and the done list.
User story	A user story is a single high-level requirement. In this project at the beginning of each iteration the next highest priority user stories are printed from the backlog onto cards and placed on the wallboard at the beginning of an iteration
Task	A portion of development or other work, such as testing or documentation, that is part of a single story. Tasks are sized so that they can be completed by a single developer in 1 to 4 hours
Burn down chart	A graph used to show the progress of story completion during an iteration. This graph is displayed on the wallboard and thrown away at the end of the iteration
Done list	A list of criteria that a story must meet before it is considered finalised. This is displayed on the wallboard
Whiteboard	A normal whiteboard used for discussion and design sessions
Working software	A semi-complete executable version of the system under development provided to the engineers on their desktops in a beta environment so they could perform UAT whenever they had time
Wiki	An electronic storage place for project documents. All documents on the wiki are available to the project team

Boundary spanning activities

A boundary spanning activity is performed to enable reciprocal interaction between project team members and external parties to the project to meet project goals (Levina & Vaast, 2005). Boundary spanning activities were plentiful in Storm and are described in Table 29. Boundary spanning was needed to coordinate with other business units, other organisations, but the bulk of this coordination was with the engineers. There were about 80 engineers, although they were never all present together as they worked on shifts. This made it difficult for a developer to approach the same engineer twice about a problem in a short time frame. After some time, the tester partially resolved this problem by selecting four engineers to be beta testers who acted as preferred contact points for the project team. These selected engineers then provided most of the feedback on new versions, carried out beta-testing and provided requirements and design detail clarification. Occasionally a critical international event occurred and in this situation, no engineer was available because they all had to focus exclusively on their work.

Table 29 Storm boundary-spanning activities

Primary Coordination mechanism	Frequency	How this coordination mechanism was used in Storm
Workshop to generate backlog	Per project	Only used on one sub-project. This involved working with the customer (a manager) to identify and prioritise user stories to generate a backlog
Nominated specialist end-user	Per project	A group of four volunteers from the engineers group were selected as beta testers and to act as a main contact for the developers and tester on requirements and system feedback
User story prioritisation	Per iteration	A session with the whole team when stories are prioritised based on business needs and technical constraints
Software demonstration	Per iteration	A session when the current version of the software is shown to the engineers
Dedicated time period for consultation	Daily	A regular period of time during each day when engineers are known to be available for consultation by the project team
Formal meeting	Ad hoc	A formal pre-arranged meeting with an external business unit to discuss important project problems or needs
Informal negotiation f2f	Ad hoc	An informal meeting with one or more other business units or external organisations to discuss project problems or needs
Breakdown session (ad hoc)	Ad hoc	A session with whole or part of the project team and the engineers to discuss the detailed requirements of a story
Acquisition of specialist knowledge	Ad hoc	When a project team member is sent out to another team to spend some time with them to learn about their system. This was necessary because the new system and the external system would be integrated in the future

The way the team coordinated with the engineers followed the same pattern as the intra-team coordination. Boundary spanning activities occurred at different frequencies: per project, per iteration, daily, and ad hoc. Boundary spanning differed from synchronisation activities in two ways. Synchronisation activities involved the whole team or large subsets of the team interacting with each other. Boundary spanning more commonly involved interaction between individuals in the team with external parties (i.e. with other teams in the organisation such as the engineers, other business units, or other organisations). The daily contact was possible because all of the engineers had a free period each afternoon for thinking, creating, and problem solving. The development team came to recognise this as a good time to approach the engineers, and in this way, they developed a daily interaction with that group.

Boundary spanning artefacts

There was only a single artefact used for boundary spanning purposes. The tester would develop a simple list of tests, User Acceptance Tests, and give it to the engineers:

"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." [CT04]

Proximity

In Storm, the team of 10, which included the project manager and the tester, were in close proximity because they all occupied a single large project room without partitions. They could all see each other and the wallboard and other whiteboards with information on the walls of the room. When the project began the project team members were all external contractors and the project manager negotiated with the organisation that they should work in this way.

Availability

In Storm, the team all worked full time on the project and were constantly available to one another. This was also negotiated between the project manager and the organisation when the project began.

Substitutability

In Storm substitutability was mixed. Most of the team were hired for their Java skills, but there was some role division. A tester was hired after a need was seen for such a role, there was a project leader, and two domain experts who joined the team from other organisational units to help with understanding the existing mainframe being replaced. Substitutability did occur, all the Java developers could take on each other's work, and developers carried out testing tasks on behalf of the tester. They were able to do this because test tasks were displayed on the wallboard and developers could see which tasks were holding up completion of a story:

"So a developer ... if they see a couple of test tasks hanging on there for a good couple of days, they may just say "[Sam, the tester] do you want me to do these, you look pretty busy there", and they'll go off and do them." [CT04]

Furthermore, each developer had the same high level of Java skills and could self-select any task from the wallboard to work on:

"You are meant to go grab the next highest priority one [story card] off the board, but generally we aren't quite that strict about it, we all know, since we are in a team of four people or five people, what each person is going to be best tackling. And you kind of just go 'yes' I will do that, and no one argues. We just know which ones to grab next." [CT01]

Coordinator role

A *coordinator role* is a coordination mechanism apparent in Storm, that did not occur in Land. A coordinator role is the role taken by a person on a development team who initiates and sustains reciprocal interaction with other external parties to the project to meet project goals. This person acts as a primary point of contact between the project team members and other organisational units or external parties.

Storm had two different coordinator roles. The project manager acted as a coordinator between the organisation management and the project team:

"[the project manager] just kind of guides us and we have got him to shield us from making you know, we can just give him the information we need him to make a decision on, or get more information from other people on."
[CT01]

The tester acted as a coordinator between the engineers and the project team:

"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." [CT04]

The complexity of the project made these coordinator roles necessary. As discussed in the literature review, according to Xia and Lee (2005), a project is complex when it includes multiple user units, software environments, technology platforms, a lot of integration with other systems, and real-time data processing (see Table 6). All of these factors were present in Storm. To reduce the need for all of the project team members to be involved in handling these complexities, these coordinator roles were initiated as the project progressed.

STORM COORDINATION STRATEGY

Coordination mechanisms and dependencies occurring in Storm are shown in Table 30. This table shows the primary and secondary dependencies, primary and secondary coordination mechanisms, and unmanaged dependencies. Columns show dependencies and rows show coordination mechanisms. Each coloured cell indicates a dependency: coordination mechanism pair. A pair is one or more instances of a coordination mechanism and its associated dependency identified in the case data.

Table 30 Storm dependencies and coordination mechanisms

Key for table			Dependencies								
			Knowledge				Task	Resource			
								Entity	Technical		
			Resource dependency	Task dependency	Knowledge dependency	Unmanaged dependency	Expertise	Requirements	Task allocation	Historical	Activity
Coordination Mechanisms	Synchronisation activities	Domain specialist on team									
		Assignment of story									
		Breakdown session (iteration)									
		Daily stand-up									
		Cross-team talk									
		Pair program									
		Breakdown session (ad hoc)									
		Informal negotiation f2f									
		Continuous build									
	Synchronisation artefacts	Product backlog									
		Wallboard									
		User story									
		Task									
		Burn down chart									
		Done list									
		Whiteboard									
		Working software									
		Wiki									
	Boundary spanning activity	Workshop to generate backlog									
		Nominated specialist end user									
		User story prioritisation									
		Software demo to user									
		Dedicated time for consult									
		Formal meeting									
		Informal negotiation f2f									
		Acquisition of specialist knowledge									
	Boundary spanning artefact	List of tests									
	Availability	Single priority team									
Proximity	Team member co-location										
	Customer co-location										
Substitutability	Redundant skill										
Coordinator role	Tester										
	Project manager										
	Unmanaged dependency										

A dependency-based view of the data in Table 30 provides the cell count data shown in Table 31 and Table 32.

Table 31 Storm dependency percentages - secondary dependencies

Secondary dependency	Count	Percentage (n=53)
Knowledge	37	70%
Task	10	19%
Resource	6	11%
TOTAL CM: dependency pairs	53	100%

Table 32 Storm dependency percentages - primary dependencies

Primary dependency	Count	Percentage (n=53)
Requirement	21	40%
Activity	10	19%
Expertise	8	15%
Task allocation	6	11%
Technical	5	9%
Historical	2	4%
Entity	1	2%
TOTAL CM: dependency pairs	53	100%

This view indicates that 70% of pairs in Storm are associated with a knowledge dependency. That is, attending to requirement dependencies, expertise dependencies, historical, and task allocation dependencies constituted most of the coordination work in Storm. Of those knowledge pairs, 57% are associated with requirements. The requirement dependency involved 40% of all pairs in the analysis.

The next most frequent group is the activity dependency at 19% of all pairs in the analysis. Expertise (15%), task allocation (11%), and technical (9%) dependencies are present, but at lower levels.

A coordination mechanism-based view provides the cell count data shown in Table 33.

Table 33 Storm coordination mechanism percentages

Secondary coordination mechanism	Count	Percentage (n=53)
Synchronisation activities	14	26%
Synchronisation artefacts	17	32%
Boundary spanning activities	12	23%
Boundary spanning artefacts	1	2%
Availability	1	2%
Proximity	4	8%
Substitutability	1	2%
Coordinator role	3	6%
TOTAL CM: dependency pairs	53	~100%

This view indicates that synchronisation activities and the artefacts produced and used during those activities are the predominant form of coordination because 58% of all pairs identified are for this purpose.

This view also indicates that, although there are 33 individual coordination mechanisms in Storm, 22 (about 66%) of them address only a single dependency.

Of the coordination mechanisms that address multiple dependencies, the wallboard, user stories, and informal face-to-face negotiation with people outside the project all addressed four dependencies simultaneously. Another group of coordination mechanisms addressed three dependencies simultaneously, breakdown sessions, cross-team talk, and team member co-location.

In summary, Storm used a strategy, shown in diagram form in Figure 13, made up of the following coordination mechanisms:

- Synchronisation activities at different frequencies: per project, iteration, daily, and ad hoc to maintain a common understanding of the status of the project among project team members. Project complexity led to a change in the size of time boxes (the duration of the iteration). In Figure 13 this is shown with an arrow pointing from the box labeled project complexity to synchronisation activity and artefacts. This is to indicate that project complexity influenced synchronisation activities and their associated artefacts
- Synchronisation artefacts created or used during those synchronisation activities. This is shown in Figure 13 with a double headed arrow between synchronisation activities and artefacts. This is to indicate that synchronisation activities produce artefacts, and those artefacts are used as the project progresses (e.g. stories placed in a shared document, then later displayed as cards on a wallboard, and finally discarded once the story was implemented).
- Boundary spanning activities performed at different frequencies: per project, iteration, daily, and ad hoc. Most of the boundary spanning activities involved project team members interacting with the engineers to achieve a steady stream of requirement details for the developers to work on, and to give feedback to the developers on the quality of the working software
- Boundary spanning artefacts created for coordinating with external parties. This is shown in Figure 13 with an arrow between boundary spanning activities and artefacts. This is to indicate that boundary spanning activities produce artefacts (e.g. request forms for external parties).

- Availability of team members, which was achieved because all project team members worked on the single project and were available simultaneously.
- Proximity of project team members, which was achieved with full co-location of the project team in a single project room.
- Substitutability of team members, which was achieved because they had overlapping skills and could perform many of each other's tasks to maintain the project schedule
- Coordinator roles, which were performed by the project manager who acted as an interface between the team and the organisation, and the tester who acted as an interface between the team and the engineers. These roles were necessary because of the complexity of the project. In Figure 13 this is shown with an arrow pointing from the box labeled project complexity to coordinator role. This is to indicate that project complexity influenced, or led to the need for a coordinator role

The coordination strategy in Storm focused on addressing requirement dependencies. Synchronisation activities and associated artefacts were the primary coordination mechanisms used to manage requirement dependencies. Availability and proximity were readily achieved because the team was made up of contractors dedicated to the single project goal and they were provided with a single project room to work in. Project team members were hired because of their high level of Java skills and this ensured substitutability. In Storm, because the end-users were not part of the team, a number of boundary spanning activities were employed to maintain a steady inward flow of requirements and feedback. Two coordinator roles were apparent in the way the project was organised: the project manager acted as a coordinator between the project team and the organisational management, and the tester acted as a coordinator between the project team and the end-users.

Figure 13 Storm coordination strategy

4.5 CASE SILVER

Project Silver took place in a small, privately owned software development company specialising in creating software products for Government and quasi-government agencies. The project was to build a new system to replace an outdated and inadequate records management system for a client. This client was a not-for-profit organisation providing services to the Health sector.

Because the five developers on the team were all new to agile methods, when the project began a consultant Scrum coach was employed. This coach assisted them for most of the project. 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 to use Microsoft.Net technologies for the development, 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.

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. This 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.

The project had few stakeholders. Silver project team consisted of four full-time permanent employees; one was the Technical Lead. The Scrum Coach/Project manager was on contract and worked 3-days a week on the project. Management consisted of three directors who were all partners in the business. Technical support staff consisted of 14 permanent employees providing IT support for client projects and for the Silver project team. This group worked in a room adjacent to the Silver project room.

There was an external client organisation paying for and using the new system. Client representatives consisted of three people from the client organisation whose role was to provide details about the system requirements and quality. One of these people was, in Scrum terminology, the designated Sponsor who was a senior manager in the client

organisation; another was the Product Owner who was a representative of the management of the client organisation. The third was a representative of the end-users. The Product Owner and representative came to the product demonstrations and sprint planning meetings at each sprint. The end-users were all located at the client site and the team estimated that 30 people used the legacy system on the client site. These stakeholders and their main communication links are shown in Figure 14.

Figure 14 Silver: stakeholders and their communication links

Table 34 shows the sources of case data.

Table 34 Silver data sources

Source	Identifying Code	Detail
Interview	DP01A DP01B DT01 DT02 DT03	Double interview with Senior Developer/Technical Lead (A) and Agile Coach/Scrum Master [B] Senior Developer Senior Developer Software Developer
Organisational website	OW	
Public document	PD	None available
Questionnaire A	QA	Project profile and software development practices
Questionnaire B	QB	Software development practices only
Project documents	CD	Full records from the Rally™ application detailing iterations and user stories (84 pages)
Photographs	PH	Stand up protocol Wallboard Done lists (2) Retrospective notes Notes about ground rules and testing
Sketches	SD	Room layout
Field notes	FN	Initial meeting with CEO, Technical Lead, and Agile Coach

Details of the case are provided in Appendix J Silver Case Description.

SILVER DEPENDENCIES

Managed dependencies in Silver

Analysis of the data gathered for project Silver shows that the project had four *knowledge* dependencies: requirement, expertise, task allocation, and historical. Silver had a single *task* dependency, which was an activity dependency. Entity and technical dependencies, forms of *resource* dependency, were both present. Table 35 provides illustrative evidence for each primary dependency and shows the grouping of primary dependencies into secondary dependencies. Dependencies were previously defined in the Land and Storm analyses, and are also provided in Appendix G Glossary of Terms.

Table 35 Silver dependencies

Secondary Dependency	Primary Dependency	Evidence	Explanation
Knowledge	Expertise	<i>"...we had effectively a day and a half when development would stop, testing would come in and everyone would test the thing to destruction effectively and the knowledge of how it worked at that point would be picked up." [DT01]</i>	The team <u>members learn about the software system</u> (dependency) by all working together on regression testing the system when they <u>prepared for the product demonstration</u> (coordination mechanism)
	Requirement	<i>"We knew from the Product Owner what they wanted, so we got together in this room, looked at the stories, discussed how we would implement, and discussed the acceptance test criteria, and went round [the table], and everyone had one story, and they would write down the acceptance test criteria that we agree on. And then, make tasks for each of those stories." [DP01B]</i>	<u>At a story breakdown session</u> (coordination mechanism), requirements from the Product Owner written on story cards were discussed among the team and acceptance tests were written for each story. Each story was broken down into tasks. In this way, team members <u>learn about the requirements</u> (dependency) and the tasks that make up each story
	Task allocation	<i>"So normally that would be the task wall you know, 'where are the avatars?' to quickly see what people are working on." [DT03]</i>	The avatars on tasks on the wallboard (coordination mechanism) show the project team who is working on what task at present (dependency).
	Historical	<i>"What do you put into Rally?" [Researcher] "... all the user stories in the Product Backlog, all the details about the user stories, which would be a short description and the size, if we have it. It is also electronic backup of the task wall, because 6 months now I will have forgotten velocity of Sprint 3." [DP01B]</i>	<u>Rally</u> , acts as a repository for project information (coordination mechanism) <u>so that past decisions are available</u> (dependency)
Task	Activity	<i>"Well, ..., it became a discovery process, if we didn't know, we were ignorant of the possibility of these dependencies, until we found that somebody would be working on a story at..., the top of the board that affected, or was required by, a story below it, so the next person, well occasionally we started off trying to work on the second story and then found out 'hold on, I can't complete this because it requires something that you are working on'." [DT03]</i>	<u>The wallboard</u> displaying stories and tasks (coordination mechanism) acted as an indicator of potential <u>activity</u> timing conflicts (dependency)
Resource	Entity	<i>"They are really nice, and really helpful but and, you walk over and say 'can you help me with this?'" [DP01B]</i>	<u>The IT support team</u> (dependency) in the next room were <u>approached directly as and when needed</u> (coordination mechanism)

Secondary Dependency	Primary Dependency	Evidence	Explanation
	Technical	<p><i>"...working in a team requires a different discipline to working alone, ... because you are interacting with other peoples' code. So having these tests in place, which basically made sure that the behaviour or the functionality of the code doesn't change. So that if someone comes along and makes a modification, that they think in their world is quite normal but in fact breaks the way that you were expecting to use that user code, then there is a little pop up. So yes, continuous integration is a really useful tool or basically these automatic testing because that's the primary thing it does is it will automate the execution of the unit tests instead of going for a whole day and then deciding to run them." [DT03]</i></p>	<p><u>The continuous integration and test system</u> (coordination mechanism) provides prompt information to the development team about the <u>impact of code changes</u> (dependency)</p>

Unmanaged dependencies in Silver

Silver had two unmanaged dependencies, *requirement* and *activity*. Unmanaged requirement dependencies occurred when direct communication between the project team members and the end-users of the new system was blocked, and slow response on requests for information about requirements from the client representatives held up progress.

"Ok and I had, in my calendar, every two days, follow up [Nicole], follow up with Nicole, 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 that would be notes, nag Nicole, nag Nicole, nag Nicole." [DP01B]

The project team devised a system for putting scheduled tasks 'on hold' while waiting for responses from the client. They would then move on to address other tasks or stories. They also occasionally removed a story from a sprint when the story could not be broken down into tasks, or the detail of a task was unclear. This happened because the client was unable to provide details about the story or task in time for the project team to maintain their schedule. Although the project was not unduly held up by this lack of information, it meant unexpected task switching had to occur and partially completed development work had to be put on hold.

Another unmanaged activity dependency occurred when the rule of having a maximum of three stories open at one time caused progress to slow. This rule was put in place to ensure that at least one or two stories were complete (developed and tested) at the end of the sprint. Sometimes dependencies between stories conflicted with this rule and work was held up, as two of the developers explain:

"...we realised that some of these stories have dependencies between each other, and tried to make sure that we knew as we were going through the process of documenting what their Done state was, and what tasks were involved, figuring out that these ones are actually related. They are quite closely coupled and just trying to make sure that we didn't get into a state where we had people working on all of the stories, all the ones we had defined. Because remember, we tried to work on only one or two stories at a time....I think we did get up to three and the reason for that is to make sure that you don't have too many things that are untested. So if you decided you had enough people and decided to start on all of the stories at once and none of them got finished, then at the end of the sprint you would have nothing. If you, instead, put those resources into working on one thing and then tested one thing, then you are more likely to get more of those stories actually finished instead of having them left open at the end of the Sprint. So given that constraint of not having too many stories open at the same time, these dependencies basically ended up blocking anymore work being done." [DT03]

"The ruling about only having three user stories open at once was a little restrictive because especially towards the end of each, when there was only

testing left, only one person should be doing the testing otherwise you're duplicating effort and it's a waste of time. So we did tend to find ourselves falling over each other a bit because of that." [DT01]

This situation shows that the coordination mechanisms, the stories and the rule about three stories being open at once, led to a dependency later in the sprint. This shows that the relationship between coordination mechanisms and dependencies is not simple. In this project, coordination mechanisms addressing dependencies at one stage of the project seemed to cause dependencies later in the project.

Summary of dependencies in Silver

Table 36 summarises the dependencies identified in Silver. Storm had the same dependencies.

Table 36 Silver summary of dependencies

Secondary dependencies	Knowledge	Task	Resource
Primary dependencies	Requirements Expertise Task allocation Historical	Activity	Technical Entity

SILVER COORDINATION MECHANISMS

Silver, like the other projects, used various coordination mechanisms to address project dependencies. Numerous primary coordination mechanisms were identifiable, and were grouped into the following categories: synchronisation activities, synchronisation artefacts, boundary spanning activities, boundary spanning artefacts, proximity, and substitutability.

Synchronisation activities

Synchronisation activities occurred at different frequencies as shown in Table 37.

Table 37 Silver synchronisation activities

Primary Coordination mechanism	Frequency	How this coordination mechanism was used in Silver
Initial infrastructure development sprint	Per project	The initial phase of the project when the Technical Lead prepared basic system architecture (including a three-layered architecture, generic classes, and security components) as proof of concept and to provide an initial framework to structure the development work
Sprint planning session/ Breakdown session	Per iteration	User stories for the iteration were selected from the backlog and manual acceptance tests written for each story, stories were decomposed into tasks, then each story and its tasks are placed on the wallboard in order of story priority
Two-week sprint	Per iteration	A two-week time box beginning on Tuesday and finishing on Monday
Retrospective	Per iteration	A whole-team session held at the end of each sprint for discussing what worked well and what did not work well during the sprint. New practices were discussed that could be trialled in the next sprint
Release of software to end-user	Per iteration	The software was prepared for release to the client test system at the end of each sprint. End-users were to trial the software and provide feedback to the team. Although feedback was not given, the team maintained the practice during the project
Daily stand-up	Daily	A short whole-team meeting held each morning at 10 am. Each team member stated, 'what I did yesterday, what I plan to do today, what are my impediments'
Cross-team talk	Ad hoc	Informal talk, when project team members ask questions of everyone else in the project room, and all the team members can overhear the question, the response and any other conversation about the project
Pair program/ Help session	Ad hoc	Formal pair programming was not used but developers would sit together to help each other if it was requested
Broadcast email	Ad hoc	If an issue was considered serious, and it was important to ensure information was received by all the team, then a developer would send an email notification to everyone on the team
Preparation for product demonstration	Ad hoc	Each Thursday morning a developer was randomly selected to demonstrate the product to the client on the following Tuesday. This involved understanding all of the stories competed in the sprint, performing regression testing, ensuring bugs were noted and resolved, writing a 'script' to follow during the demonstration, and preparing suitable test data for the demonstration
Product backlog maintenance session	Ad hoc	A backlog is a prioritised list of epics and stories to be addressed. At random intervals, usually about twice per sprint, the team would sit and discuss stories and size them. The team maintained the backlog so that it held enough stories for one to one and a half sprints
Informal negotiation f2f	Ad hoc	This was any discussion or negotiation that occurred within the team
Manual acceptance test	Ad hoc	This is story-level acceptance testing. An acceptance test is written for each story at the breakdown session. After the code for the story was written, it was manually checked against the test criteria by another developer on the team. If the software

Primary Coordination mechanism	Frequency	How this coordination mechanism was used in Silver
		failed, a pink 'bug' note went on the wallboard and the developer who wrote the code would fix the bug
Continuous integration and test (build)	Ad hoc	Continuous integration involves continual 'builds' (integration and execution) of the system components. If the build 'fails' then there is a bug or other problem with the code base that must be resolved

Synchronisation artefacts

In Silver, synchronisation artefacts were either publicly visible or invisible but available to all of the team. Visible artefacts were the wallboard, user stories and tasks, a burn down chart, avatars, the Done list, and ground rules. All visible artefacts were displayed on a single wall and were clearly visible to all team members from their desks. Invisible artefacts were the code standards, working software, the Rally software repository, the three-layered architecture, the source code control system (Subversion), and the unit tests. The synchronisation artefacts, and an explanation of their use, are shown in Table 38.

Boundary spanning activities

Boundary spanning activities were few in Silver because of the lack of access to the end-users and the strictly scheduled meetings with the client representatives. Interactions with the client representatives were restricted to once each sprint when they attended the product demonstration sessions, and immediately following this they attended an initial sprint planning session to discuss and prioritise stories. Opportunities for project team members to interact with the client representatives or the end-users, in an ad hoc or unscheduled manner, were infrequent. This was a major problem in the project and caused various issues, as described in the introduction to this case. This lack of contact increased project uncertainty about the details of requirements. To cope with this, synchronisation artefacts were adjusted because the team members changed the process to include a 'blocked' column on the wallboard for displaying tasks halted waiting for client feedback. Another adjustment was to de-prioritised stories and remove them from the sprint, which meant that either less work was addressed during that sprint, or replacement stories were selected, so as not to delay the sprint.

The Technical Lead and the Scrum Coach discussed the outcomes of this uncertainty:

*"...having to guess a lot more than we would have liked to have done, and not really getting concrete confirmation of those guesses by the end users."
[DP01A]*

“It sometimes changed our priorities...so 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.” [DP01B]

Table 38 Silver synchronisation artefacts

Primary Coordination mechanism	How this coordination mechanism was used in Silver
Wallboard	A whiteboard, clearly visible to all of the team, on the wall in the project room displaying current work in the form of stories, tasks, the progress of tasks, the allocation of tasks, the burn down chart, and the done list.
User story	A user story is a single high-level requirement. In this project at the beginning of each iteration the next highest priority user stories are printed from the backlog onto cards and placed on the wallboard at the beginning of an iteration
Task	A portion of development or other work, such as testing or documentation, that is part of a single story. Tasks are sized so that they can be completed by a single developer in 1 to 4 hours
Avatar on task	This small caricature of a developer was placed on a task to identify who was currently working on it
Burn down chart	A graph used to show the progress of story completion during an iteration. This graph is displayed on the wallboard and thrown away at the end of the iteration
Done list	A list of criteria that a story must meet before it is considered finalised. This is displayed on the wall beside the wallboard
Ground rule	A list of rules created during the retrospective sessions. The list of rules was displayed on the wall beside the wallboard
Working software	A semi-complete executable version of the system under development provided to the end-users at the client site, on their desktops in a beta environment, so they could try out the software whenever they had time
Rally	A software application designed to support the Scrum methodology. Acts as a repository for storing backlog, stories, tasks, progress information and for generating burn down charts
Code standards	Accepted ways to write code that makes it easier for others in the project team to read and understand each other’s code
Source code control (Subversion)	A software application for controlling access to the source code, and code versioning. This software allows two or more developers to work on the same code simultaneously. Changes are merged when the code is checked in to the application
Layered architecture	The system had a three-layered architecture: a GUI layer, a business layer, and a data access layer. This means developers could work simultaneously on the same portion of functionality but in different layers. This sped up development
Unit test suite	A set of tests kept and re-run at frequent intervals whenever new code is added to the system. New unit tests are added to the suite as the new code is written

The project team had one other external group to interact with, the IT support team in their own organisation, who were easily accessible and located in the adjacent room.

Boundary spanning artefacts

There was only a single artefact used for boundary spanning: the working software provided to the end users. Since this group did not provide feedback on the software, this was not an effective means of gaining feedback in this project.

Proximity

In Silver, the team, along with the Scrum Coach and the tester, were all in the same project room together with their desks arranged to face the wallboard. This close proximity meant that the task allocation dependency (who is doing what and when) and the historical knowledge dependency (knowledge about past decisions) were easily addressed:

"Just ask the room, and either someone will remember something, or maybe just know where to go and look, and it might be at times having to go and read the code to understand what it's doing to find out what the decisions were." [DT03]

Availability

In Silver, the team all worked full time on the project and were constantly available to one another. The Scrum Coach worked three days per week on the project. Availability was not remarked on by anyone working on the project because it did not cause any problems, everyone was always available. The effect of this constant availability was the same as that for proximity, that is, when a team member needed a fast response to their query involving expertise, task allocation, or historical information, another team member could immediately respond to their request.

Substitutability

The coordination mechanism '*redundant skill*' did not arise naturally in the project, as initially the skill sets of the developers varied. The Scrum Coach made an effort to ensure that skills were shared, and help was provided to up-skill individual team members:

"... in the beginning, [Sam], who did not know C#, and I had a conversation with him, about how he did not feel comfortable. So, I told him I basically didn't care, I was happy to take him on from a project management perspective. 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, and testing, and making user stories." [DP01 B]

"And basically everyone was considered equal; they can work on anything." [DT03]

SILVER COORDINATION STRATEGY

Coordination mechanisms and dependencies in Silver are shown in Table 39. This table shows the primary and secondary dependencies, primary and secondary coordination mechanisms, and unmanaged dependencies. Columns show dependencies and rows show coordination mechanisms. Each coloured cell indicates a dependency: coordination mechanism pair. Each pair indicates one or more instances of a coordination mechanism and its associated dependency identified in the case data.

Table 39 Silver dependencies and coordination mechanisms

Key for table			Dependencies							
			Knowledge				Task	Resource		
								Entity	Technical	
			Resource dependency			Expertise	Requirements	Task allocation	Historical	Activity
Coordination Mechanisms	Synchronisation activities	Initial infrastructure sprint								
		Sprint planning/breakdown session								
		Two-week sprint								
		Retrospective								
		Release of software								
		Daily stand-up								
		Cross-team talk								
		Pair program/help session								
		Broadcast email								
		Preparation for product demo								
		Product backlog maintenance								
		Informal negotiation f2f								
		Manual acceptance test								
		Continuous integration & test								
	Synchronisation artefacts	Code standards								
		Wallboard								
		Avatar on task								
		User story								
		Task								
		Rally								
		Layered architecture								
		Source code control								
		Unit test suite								
		Working software								
		Done list								
		Ground rule								
		Boundary spanning activity	User story prioritisation							
			Software demo to client							
	Informal negotiation f2f									
	Proximity	Team member co-location								
	Availability	Full-time team								
	Substitutability	Redundant skill								
		Unmanaged dependency								

A dependency-based view of the data in Table 39 provides the cell count data shown in Table 40 and Table 41.

Table 40 Silver dependency percentages - secondary dependencies

Secondary dependency	Count	Percentage (n=44)
Knowledge	33	75%
Task	7	16%
Resource	4	9%
TOTAL CM: dependency pairs	44	~100%

Table 41 Silver dependency percentages - primary dependencies

Primary dependency	Count	Percentage (n=44)
Requirement	8	18%
Activity	7	16%
Expertise	10	23%
Task allocation	6	14%
Technical	3	7%
Historical	9	20%
Entity	1	2%
TOTAL CM: dependency pairs	44	~100%

This view indicates 75% of pairs in Silver were associated with a knowledge dependency. That is, attending to requirement dependencies, expertise dependencies, historical, and task allocation dependencies constituted most of the coordination work in Silver. Of those knowledge pairs most involved expertise (23%). Within the different knowledge dependencies, no primary dependency stands out. The relative frequency of the four types shows they are all contributing: expertise at 23%, historical 20%, requirement 18%, and task allocation 14%.

Activity dependencies were noteworthy at 16%, but technical and entity dependencies were at very lower levels.

A coordination mechanism-based view provides the cell count data shown in Table 42.

Table 42 Silver coordination mechanism percentages

Secondary coordination mechanism	Count	Percentage (n=44)
Synchronisation activities	18	41%
Synchronisation artefacts	18	41%
Boundary spanning activities	3	7%
Availability	3	5%
Proximity	2	5%
Substitutability	1	2%
TOTAL CM: dependency pairs	44	~100%

This view indicates that synchronisation activities, and the artefacts produced and used during those activities, were the predominant form of coordination mechanism because 82% of all pairs were for this purpose.

This view also indicates that although Silver used 44 individual coordination mechanisms, half of them (50%) addressed only a single dependency. Of those coordination mechanisms addressing multiple dependencies, the wallboard addressed three dependencies simultaneously; the remainder of the coordination mechanisms addressed two dependencies.

In summary, Silver used a strategy made up of the following coordination mechanisms, illustrated in Figure 15:

- Synchronisation activities at different frequencies per project, iteration, daily, and ad hoc
- Synchronisation artefacts created or used during those synchronisation activities. This is shown in Figure 15 with a double headed arrow between synchronisation activities and artefacts. This indicates that synchronisation activities produce artefacts, and those artefacts are used as the project progresses
- Project uncertainty affected synchronisation activities, and production and use of synchronisation artefacts. To maintain the iteration length when requirements were uncertain (i.e. under conditions of project uncertainty), the team used a task switching process that involved dropping stories or tasks, and beginning work on alternative stories. In addition, artefacts such as a 'blocked' column on the wallboard were introduced to cope with project uncertainty. In Figure 15 this is shown with an arrow pointing from the box labeled project uncertainty to synchronisation activity and synchronisation artifact.
- Availability of team members, which was achieved because all project team members worked on the single project and were available simultaneously
- Proximity of project team members, which was achieved by locating all of the project team in a single project room
- Substitutability of team members, which was achieved because they had overlapping skills and could perform many of each other's tasks to maintain the schedule
- Boundary spanning activities performed at two frequencies: per iteration and ad hoc

Figure 15 Silver coordination strategy

4.6 CASE ROCK

Project Rock occurred in a retail bank. The project involved the complete redevelopment of an on-line application to enable clients to view and manage their credit card statements online. The new system needed to work seamlessly with existing on-line and off-line banking functions, as well as systems external to the bank, including a mailing company, a mobile alert service company, a credit card company, and an international airline.

The project was not critical to the functioning of the bank, but it was important. The new system would streamline a customer's on-line banking experience, and help to maintain the bank's competitiveness with similar banks.

The project began in early 2009. A business analyst was employed to produce the high-level scope for the project, and in July 2010, a project team was formed. This team fluctuated in size over the life of the project. The project had seven core full-time employees consisting of a project manager, a business analyst, a Host¹² specialist, two testers, and two developers. In the final phase of the project this team size rose to include 15 people. It was particularly notable that people joined the project with particular specialisations (e.g. tester or Host specialist) and did not move outside of those roles.

The business analyst described the development method as "waterfall on the wall." The method was a hybrid, including daily stand-up meetings, the production control method called Kanban, use cases for object-oriented design, and traditional roles used in the SDLC methodology for the project team members. The method had no iterations or sprints, instead using a Kanban wallboard displaying a continuous stream of work. The overall organisation had adopted Scrum while this project was in progress, and the Rock project team was located in one section of an open-plan floor of a building, alongside eight other project teams using Scrum.

The project had some problems. They included slow response on requirements from external organisations, accommodating the fluctuating nature of personnel assigned to the project, post release problems caused by unrecognised data complexity, sharing a test data bank, and negotiating the new design for the system with the stakeholders. The expected delivery date was compromised, but this was caused by problems external to the project, including problems within the bank, and when the Christchurch earthquake struck. When these problems halted the project, the project manager decided to use this time to incorporate additional enhancements to the new system, and the capability to manage foreign currency statements was included. On 24th March 2011, the system went live to the customers. There were some issues the week after release, involving long hours for the development team to make fixes. Nevertheless, at the time of data collection, the system was functioning successfully.

¹² The Host was the name for the mainframe computer. This central system hosted the transaction processing applications and key data.

Rock had a variety of project stakeholders. The Rock project team fluctuated in size over the life of the project, increasing in size to about 15 people for 6 to 8 weeks in the final phase of the project. The project had seven core full-time employees consisting of:

- 1 Project manager – assigned to multiple projects simultaneously
- 1 Business analyst – developed scope and requirements, and acted to address project impediments as they arose
- 1 Host specialist - a specialist in the back-end of the system who designed the Host components of the new system to fit with the existing Host applications. The Host system was a mainframe system written in Cobol.
- 2 Testers – one for Host testing, one for testing the front-end
- 2 Java developers - specialists in the front-end of the system

As the project progressed, further part-time staff were added: a Java developer, three Host developers, and a Middleware specialist. Another stakeholder group was another bank development team sharing the test environment. This team was working on a similar system and was using the same test data resource as the Rock team.

Further stakeholders included all of the people with an interest in the system. This included: the Head of Retail Banking, who acted as a proxy customer, the Head of Online Banking, the Head of Credit Card division, who also acted as a proxy customer, the Head of Legal division, the Head of Marketing and Communications, the System Architect who controlled and maintained the mainframe database structure, and the Credit Card testing group. This group interacted with CreditCo.

External stakeholders included one other major New Zealand bank, one major Australian bank, and three non-bank organisations who were (all names are changed): MailCo was a company that archived and printed statements, and sent standard letters to bank customers, AlertCo was a company that provided alerts to customers as part of the banks mobile banking service, CreditCardCo was a company that provided the credit card system. Interaction with this group was primarily for setting up and acquiring appropriate test data. AirCo was another external stakeholder. This company provided a points system related to the credit card system and wanted the bank to provide a link to their site from the new system.

Table 43 shows the sources of case data.

Table 43 Rock data sources

Source	Identifying Code	Detail
Interview	GP01 GT01 GT02 GT03	Senior Business Analyst Senior Analyst Programmer Test Analyst Technical Designer
Organisational website	OW	Organisation details
Public document	PD	None available
Questionnaire A	QA	Completed
Questionnaire B	QB	Completed
Project documents	CD	One work item card
Photographs	PH	None
Sketches	SD	Layout of wallboard
Field notes	FN	Notes taken at each of the four interviews

Details of the case are provided in Appendix L Rock Case Description.

ROCK DEPENDENCIES

Managed dependencies in Rock

Overall, Rock had a total of seven managed dependencies. Of these, four were *knowledge* dependencies: requirement, expertise, task allocation, and historical. Rock had two *task* dependencies: activity and business process. Also present were entity and technical dependencies, which are forms of *resource* dependency. Table 44 provides illustrative evidence for each primary dependency.

Decision rules for Rock dependency analysis

Classification decisions for Rock were the same as those for Storm and Silver. One difference was that defects and data resources were mentioned more often in this project than in the previous project, and decisions about their classification were made as follows:

- Software defects were classified as a form of technical dependency because a defect is similar to the case where a software component is not available.
- Problems with data and data structures were classified as a form of entity dependency because they were things the system needed to function.

Table 44 Rock dependencies

Secondary Dependency	Primary Dependency	Evidence	Explanation
Knowledge	Expertise	<i>"So my main contact was with middleware for example. So if I had an issue or a problem then I would go straight to – he was just sitting right in front of me, [Valerie] was, so I would talk with him. And [Kate] was also on my left. So [Kate] was from the Host. So we did have those conversations." [GT01]</i>	The team <u>members learn about the software system</u> (dependency) by asking the appropriate specialist on the project team who was <u>sitting nearby</u> (coordination mechanism)
	Requirement	<i>"I think being a tester I want to know what everything looks like. So I'm not sure what the developers' mindset is, they get a whole lot of that you've got to do this, this, this and this and they develop a code around that. For my mind I would like to know what you are using that for." [GT02] "So how did you find that out? That as a tester you needed to know that so how did you find that out?" [Researcher] "It's in the use case. The front end is in the use case and the host stuff is in the system design." [GT02]</i>	<u>The tester needs to know about the requirements</u> (dependency) and finds out from studying <u>the use case document</u> (coordination mechanism)
	Task allocation	<i>"So the stand up meeting I think contributed a lot to it. Because you'd know during those meetings what people were working on, and who was working on what and so on." [GT01]</i>	Information about <u>who was doing what</u> (dependency) is learnt at the <u>stand-up meeting</u> (coordination mechanism)
	Historical	<i>"What we do is, I don't normally just go on my own because [Mark] is also quite new as well to the whole project. I think he also joined in halfway through, so for the benefit of everyone, usually [the project manager] will [ask] the organisation to have a phone conference with anyone that we want them to get involved. Because I think in the past there are only just probably three of them that are heavily involved in the discussion. So we just pull them this three back to the meeting and then we just ask questions. And then after that once we get what we got and we are still having any doubts, I will then go and get them to talk to them." [GT03]</i>	<u>Knowledge about past decisions</u> (dependency) is acquired at <u>a phone conference with stakeholders</u> (coordination mechanism)
Task	Activity	<i>"So we had work in progress limits. Based on the number of people available to do things but then you would start doing something and then be waiting for some more information to come from somebody so you would have something else to go</i>	<u>Work in progress limits</u> (coordination mechanism) were set to ensure that <u>one activity did not halt progress</u> (dependency). The project member could switch tasks and continue working on another related work item

Secondary Dependency	Primary Dependency	Evidence	Explanation
		<i>on with. In the development again, I think it was two for each. And then testing it was one thing at a time it was supposed to be. And they were guidelines for how many things could be in each box at any time. The integration was basically, once they'd been extended unit tested they could pile up in the integration column so it didn't have a work in progress number." [GT02]</i>	
Resource	Entity	<i>"Well, we wanted Host development and we didn't have enough. And then maybe we got a whole bunch of host developers and then for a couple of weeks or something they were whipped away with work in flight." [GP01]</i>	<u>People with relevant expertise</u> (dependency) were not <u>consistently available</u> (coordination mechanism) to complete the work required
	Technical	<i>"At the start we'd have – so you'd have one first initial build, and then once testing is started, the daily builds would pile up. So any time you checked in code, it would trigger off a build of the code, and at 5am in the morning, I think, it would deploy to test. But you also had some instances where you needed the change right away, because the testers wanted it right away. So you could do a lunch time build, so you check it in, do a manual run of the job to build it and to deploy it." [GT01]</i>	<u>Continuous integration</u> (coordination mechanism) ensured the current version of the system was <u>fully functioning</u> and there were no technical faults (dependency)

Unmanaged dependencies in Rock

Rock had unmanaged expertise, requirements, historical, business process, and entity dependencies. All of these dependencies were caused by factors external to the project, that is, other business units or external parties were inadequate in their response to requests for resources or information. In some cases, existing business process schedules within the wider organisation had to be accommodated by the project team members causing them to wait, and affecting project progress.

Expertise dependencies were unmanaged when people with appropriate skills were needed to complete work, but they were not available:

"Well, we wanted Host development and we didn't have enough. And then maybe we got a whole bunch of Host developers, and then for a couple of weeks or something they were whipped away with work in flight." [GP01]

Requirements dependencies were unmanaged when information on requirements was not readily available, and involved the business analyst locating appropriate people for their input:

"[I] waited for some of the requirements to get clarified if it came up...for various questions for the business, and if the business analyst couldn't answer it on the spot, they had to go out and find who to ask." [GP02]

Historical dependencies were unmanaged when the information about the existing system that the new system had to integrate with were unknown to anyone on the development team. Knowledge about such issues was sometimes available but there was no specific mechanism for acquiring this knowledge:

"Because something won't work and nobody can figure out why and then somebody, somewhere will say, "Oh there was a production defect," or, "We had to make a fix four years ago." [GT02]

Business process dependencies were unmanaged when interacting with external parties. The business analyst explained:

"I waited for [MailCo] quite a bit. They were the main waiting point." [GP01]

"And the other one was the credit card system that we were, for the initial part of the testing, reliant on when the batches were run, so I dealt with the credit card testing people...we had to run a separate batch from the test batches that they set up a year beforehand. Just to get some data to come across so we could test it as it went past." [GP01]

Entity dependencies were unmanaged when things were not available when needed. These things were provided either by other business units in the organisation or by external parties. Examples included waiting for system space, and errors in the data provided by the Host system. An unscheduled change moratorium also occurred due to

the Christchurch earthquake in New Zealand. These problems halted the project. The solution was to begin a new sub-project, providing functionality for on-line foreign currency statements, and proceeding with that. The technical designer explained:

"Yes, because we are all ready to go virtually. We are all ready to go before Christmas but because of the change moratorium and the space – that was the first delay that the [other bank] is not allowing us to go in because they are not signing off the space allocated to us." [GT03]

"...and because we are wanting to do a lot of things where we need to get a sign off...So these are the things that delay the whole process a lot." [GT03]

"So we had to do the data fix because that affects our project because we are using the data...the rest of [the] contractors, they have to stop, so what [the project manager] has done was to move them ... forward on to the new project. [GT03]

Data provided by another business unit was not available in the correct form and required the project to wait until the issue was addressed:

"Some of them were data, most of them were data. So the data on the Host, I think it was, was not as we expected them to be, or there was some other sort of data that somehow didn't quite fit into what we were releasing, so there were a few problems there." [GT01]

The situation was summed up by the tester:

"The original [deadline]...yeah I can't remember what it originally was but I think the complexity of the project was a bit more than originally estimated..."

And he attributed the project complexity and extended deadline to three major problems – historical data structures, large amounts of data to work with, and a system that had to integrate with multiple existing systems both internal and external:

"So we've got history, as well as just vast amounts of data. And also touched on quite a few different systems in the bank and each of those had its own little quirks..." [GT02]

The coordination mechanisms used in the project remained unchanged when these issues arose. The coping mechanism was to extend the project deadline.

Summary of dependencies in Rock

Table 45 summarises the dependencies identified in Rock.

Table 45 Rock summary of dependencies

Secondary dependencies	Knowledge	Task	Resource
Primary dependencies	Requirements Expertise Historical	Activity Business process	Entity Technical

ROCK COORDINATION MECHANISMS

Rock, like the other projects, used various coordination mechanisms to address project dependencies. Numerous primary coordination mechanisms were identifiable, and were grouped into the following secondary categories: synchronisation activities, synchronisation artefacts, boundary spanning activities, boundary spanning artefacts, proximity, availability, substitutability, and a coordinator role. One coordination mechanism not identified in the three other cases, was the use of impersonal artefacts.

Synchronisation activities

Synchronisation activities occurred at different frequencies as shown in Table 46.

Table 46 Rock synchronisation activities

Primary Coordination mechanism	Frequency	How this coordination mechanism was used in Rock
Initial meeting	Per project	At the beginning of the project, there was only a business analyst assigned to the project. Once other people joined the team an initial meeting was held to explain the requirements and the system design to the team
Daily stand-up meeting	Daily	A short whole-team meeting held each morning at 9:15 am for about 15 minutes in front of the Kanban wall. Each team member stated what they planned to do today, what they did yesterday, and any problems they faced. This session also involved making notes of impediments and sticking them on the wall.
Software release to production	Ad hoc	The current working version of the software was released to production
SMS chat	Ad hoc	The project team members would use SMS chat to ask questions of the other project team members when they had problems or had queries
Impromptu planning session	Ad hoc	After the stand-up meetings, or at any other times during the day the team might stop to plan future work such as if functionality was ready for release
Informal negotiation f2f	Ad hoc	Queries and issues were mainly addressed between team members using face-to-face conversation
Use case breakdown session	Ad hoc	A session when the whole team would meet to work on breaking down the existing use cases into work items
Phone conference with stakeholders	Ad hoc	A meeting used to clarify requirements attended by all of the team and involving the project stakeholders who provided requirements
Continuous integration and test (build)	Ad hoc	Whenever code was checked in to the JUnit system it would trigger a build of the code, and at 5am each morning, the code would deploy to the test environment. Occasionally, at the request of the testers and when a change was needed right away, there would be a lunchtime build.
Technical specialist on team	Ad hoc	A specialist in the mainframe (the Host) was included fulltime as part of the team. This person designed the parts of the new system that interacted with the Host, provided guidance whenever required, and assisted with Host development tasks when necessary.

Synchronisation artefacts

In Rock, synchronisation artefacts were either publicly visible or invisible but available to all of the team. Publicly visible artefacts were the Kanban wallboard, work items, and avatars. All visible artefacts were displayed on a single wallboard but were not clearly visible to all team members from their desks. Invisible, but publicly available artefacts were the work in progress limits, the JIRA™ application, the architecture, ClearCase™ application, and defects. The synchronisation artefacts and an explanation of how they were used are shown in Table 47.

Table 47 Rock synchronisation artefacts

Primary Coordination mechanism	How this coordination mechanism was used in Rock
Kanban wallboard	A whiteboard, clearly visible to most of the team, positioned on the wall in the project room was used for displaying current work in the form of work items. The wallboard had columns labelled: Analysis, Development, Testing, Integration, and Done. Rows were labelled: Host, Middleware, and Frontend
Work item	A work item is a portion of functionality written on a cardboard card, called a Kanban card. Each work item had a unique ID, a description of required functionality to be implemented, and was assigned to either the front end, middleware, or Host work stream
Avatar on task	This small caricature of a developer placed on a work item to identify who was currently working on it
Work in progress limit	The maximum number of work items that could be worked on simultaneously. For example, there could only be two work items open in the Analysis phase of the Host work stream at any one time
JIRA™	JIRA™ is an application for bug tracking, issue tracking, and project management. The information in JIRA™ was a reflection of the information displayed on the wallboard and was updated to reflect the wallboard status when changes were made to the wallboard. In addition, the tester used this application to store information about defects and their related documentation. All team members and other stakeholders could view this application but not all team members could make changes to it.
Architecture	The system was designed as three components: Host (the mainframe), middleware (web services), and front end (Java). This matched the existing system architecture used by the organisation
Source code control (ClearCase™)	The application was primarily used for source code control. It allowed for code sharing and merged code on check-in

Impersonal artefact

In organisation theory there is a form of coordination named impersonal mode, which occurs when “*a codified blueprint of action is impersonally specified*” (Van de Ven, et al., 1976, p. 323). This form of coordination was discussed in the literature review. In Rock, three artefacts fit this purpose: the use case specification document, the system design specification document, and the documentation of defects. Table 48 provides descriptions of these artefacts. These artefacts were produced primarily by one project team member whose role was to produce and maintain the document. Other team members would then refer to them as and when needed. These artefacts provided permanent records of the system design and defects.

Table 48 Rock impersonal artefacts

Primary Coordination mechanism	How this coordination mechanism was used in Rock
Use case specification document	This was the requirements document. It was a word document of about 200 pages containing use cases describing the high-level functionality of the new system (that is, the functional requirements). This document was developed by the business analyst and was stored in a shared document management system available to all project team members
System design specification document	A word document describing the design of the Host portion of the new system. This document was developed by a Host technical designer and was stored in a shared document management system available to all project team members
Unit test document	A document describing all unit tests used when developing the system front end
Defect	A discrepancy between the stated requirements and the code output. The tester identified defects and either routed them to the business analyst to make changes to the requirements, or to the developers who would make adjustments to the code. These defects including their description, a screen shot and associated data sets were documented and stored in the JIRA™ application as attachments

Boundary spanning activities

In project Rock, all reported activities involving boundary spanning were achieved by informal and irregular meetings (i.e. ad hoc). For example, phone conferences were used occasionally along with face-to-face meetings:

"... for the benefit of everyone, usually [the project manager] will do the organisation to have a phone conference with anyone that we want ... to get involved. Because I think in the past there are only just probably three of them [people involved in an earlier version of the system] that are heavily involved in the discussion. So we just pull them, this three, back to the meeting and then we just ask questions." [GT03]

"No, actually, no, it [the requirements document] was built from discussions with those people. So we had the [CreditCardCo], we had meetings with those people to say what are your requirements for credit cards...Same with foreign currency team, 'what do you see happening and how do you think it should work'. And [AirCo], of course, having those sort of meetings with them." [GP01]

What this situation indicates is that when project uncertainty increased, that is, when an individual on the project team needed information beyond the requirements document or the specification document, they would ask the person with expertise in that area (e.g. for Host information they would ask the technical lead, for business rules they would ask the business analyst). If that information was not available the resident expert would then consult with people in other business units or external stakeholders to acquire the necessary information. So when project uncertainty increased this led to an increase in ad hoc boundary spanning activities.

Boundary spanning artefacts

A single artefact was used for boundary spanning, and that was a test data spreadsheet produced by the senior tester. He used this to coordinate the sharing of a data used for system testing, with another project team within the organisation.

Proximity

In Rock, project team members were all seated in a section of a single large floor of the building. This was considered very useful for coordinating the work and was commented on by a number of interviewees:

"And again, the face to face always helps. We were able to explain right away. And the turnaround was quick. They would say right away that they would work on it and what the issue was if they found out after. Same with myself. The tester was beside me as well. So you had the test analyst, you had the three components people, within those three components, right beside each other." [GT01]

"The previous time I was at [this] bank, all the testers sat together, all the developers sat on a different floor, all the BAs were in a totally different building and the communication of that sort of thing took days or weeks because you would think of something, you would enter a clarification into a clarification system and the committee would look at it and decide what was going to happen to it – it was just very bureaucratic. Whereas now you just say, "Oh come over here,...and – yeah – What's supposed to happen in this scenario? To, oh well, this is supposed to happen. And the lead developer will say "Oh well, no well it's not going to show that, it will show this." "Oh no, well that's no good. That's not what we need. We need this to happen." So the discussion would go and the decision would be made, and if necessary the requirements would be updated or the business be consulted." [GT02]

Availability

Team members are generally available when needed. Although some team members worked from home, this was on scheduled days and the team used IM chat to discuss issues.

"I would work from home for like half a day a week. And [the business analyst] would work from home one day a week, and so on." [GT01]

"And how did you communicate then? [Researcher]

"That's the instant messaging chat, that worked very well, yeah. So you'd still be in touch." [GT01]

The availability of people working on the project fluctuated, and this did impact the project. For example, when host developers were needed and were not available this meant the project progress slowed (this was mentioned above as an unmanaged expertise dependency).

Substitutability

In Rock, there was little substitution of work roles. Occasionally one person would move across role boundaries, for example the technical designer had programming experience:

"Because I'm also a developer ... so whatever the contractors cannot do, I will do it ... Let's say those contractors are all tied, I'll pick up the work and I'll just do it." [GT03]

Within a role (i.e. if there were two testers) some task switching occurred:

"Just informally. If he had a question we would each review each other's stuff and he did a little bit – when I got busy he would do a little bit of host testing as well, batch testing." [GT02]

In general, there were strict role divisions that matched the system architecture: Host, middleware, and frontend. The work itself, as displayed on the Kanban wall and the Kanban cards, was always strictly separated along these same technical divisions. When the use cases (describing the system requirements) were reorganised into work items, each item was tagged with a work stream that was Host, middleware, or frontend:

"We divvied up the work based on – because there were several components to [the] online statements. There was work to do with statement options, and the other one was statement lists. So I was doing the statement lists and [Ray] was doing the statement options. So it depends, of course, based on function within the app. And as well as, I've told you already before, which component of it, it was. So I was doing front end, if it had something to do with middleware then it would be someone else, like [Val]. Or the host would be someone else again." [GT01]

"[Max] he's the other tester. So I mainly communicate with [Mark] the Business Analyst on the requirement side, you know, "This doesn't really make sense, what do you mean by that?" and he will either find out more detail or whatever, clarify the requirements. [Max] who I was testing with, [Kate] who is the senior developer and then each of the host developers depending on what their particular function was, whether I was testing that or not. If I found something that didn't look right I would go and say, "Well I'll get" [GT02]

"Well I guess everyone has to go away to find out their own specialities like the testers, the tester will rely on my spec, whereas for me I will have to rely on [the business analysts] requirements and what is the business rules and what they want. And I think [the business analyst] would then have to rely on the business to tell him what they want, and then he will just ... tell me what the business wants. So more or less he's just getting stuff from the business where I'm just getting it off from him." [GT03]

Because of this clear demarcation of roles within the project, substitutability was not a characteristic of the project, but rather the opposite - role specialisation - was the norm.

The structure of the organisation seemed to influence proximity, availability, and substitutability. The project team was working on a single floor of a building alongside about seven other project teams who were using the Scrum methodology. Therefore, to fit in with these other projects, Rock project team was also required to be co-located and to work substantially full-time on the project. The organisation's management chose to maintain strict role specialisation within the project team rather than impose, encourage, or support substitutability.

Coordinator role

A coordinator role was apparent in Rock and different people took the role depending on their specialisation within the project. The project manager acted as a coordinator between the project and other units in the organisation who could provide useful information:

"... usually [the project manager] will do the organisation to have a phone conference with anyone that we want them to get involved." [GT03]

The business analyst acted as a coordinator between the project and the organisation management who were providing the requirements:

"Waited for some of the requirements to get clarified if it came up... for various questions for the business and if the business analyst couldn't answer it on the spot they had to go out and find who to ask." [GT02]

The technical designer acted as a coordinator between the project and the Host unit:

"Again, primarily I deal with the team. And as well as, you know which people work on which areas. But in saying that, in terms of, for example, the host is quite a big vast area of applications. So sometimes if you've got a number of accounts for example, and account types, you wouldn't know necessarily nowadays who is working on those accounts or who is the person knowledgeable in that area. So you'd ask – I would ask [Kate] who, ... on the other host teams would know about the certain area. But you'd still have a contact point. A contact person. And that was the host person on your team." [GT01]

The tester acted as a coordinator between the project and other organisational units:

"...as soon as I found there was another project I got in touch with the tester and went down and visited and found out what the scope of their project was and realised that there was going to be a conflict if we just went and..." [GT02]

"And the other one was the credit card system that we were for the initial part of the testing, reliant on when the batches were run so I dealt with the credit card testing people." [GT02]

The other team members tended to use these people as channels for gaining information and did not approach other business units or external parties directly:

"I was mainly working with [Mark] who was the business analyst of the project. Now, in terms of whether you needed a decision on something or not, [Mark] would go to someone else, which would be the actual business in Auckland, for some decisions." [GT01]

Did you mainly go through him?" [Researcher]

Yeah, I didn't go straight, no." [GT01]

"Okay, why was that?" [Researcher]

"It's just easier. I think that was the role of [Mark] to be, just I would be dealing with him. And whatever he needed to find out from someone else, he would do. I think it just kept it simple as well, that I knew to speak with [Mark]. As long as I raised the issues to him." [GT01]

These coordination roles were necessary because of the uncertainty within the team about how the new system should integrate with multiple other bank systems, both internal and external systems. Barki, Rivard and Talbot (1993) identified the number of systems a system must integrate with as a measure of the uncertainty in a project. This was discussed in the section on Factors Influencing IS Project Coordination, page 83.

ROCK COORDINATION STRATEGY

Coordination mechanisms and dependencies occurring in Rock are shown in Table 49.

This table shows the primary and secondary dependencies, primary and secondary coordination mechanisms, and unmanaged dependencies. Columns show dependencies and rows show coordination mechanisms. Each coloured cell indicates a dependency: coordination mechanism pair. Each pair indicates one or more instances of a coordination mechanism and its associated dependency identified in the case data.

Table 49 Rock dependencies and coordination mechanisms

			Dependencies							
			Knowledge				Task		Resource	
Key for table			Expertise	Requirements	Task allocation	Historical	Activity	Business process	Entity	Technical
Resource dependency										
Task dependency										
Knowledge dependency										
Unmanaged dependency										
Coordination Mechanisms	Synchronisation activities	Initial meeting								
		Daily stand up								
		Software release to production								
		Informal chat using SMS								
		Impromptu planning session								
		Informal negotiation (f2f)								
		Use case breakdown session								
		Conference call with stakeholders								
		Continuous integration and test								
		Technical specialist on team								
	Synchronisation artefacts	Kanban wall								
		Work item								
		Avatar on work item								
		Work in progress limit								
		JIRA™								
		Architecture								
		Source code control								
	Impersonal artefact	Use case specification								
		System design specification								
		Unit test document								
		Defect document								
	BS activity	Informal negotiation (f2f)								
	BS artefact	Test data spreadsheet								
	Proximity	Team member co-location								
	Availability	Full-time team								
	Substitutability	Redundant skill/specialisation								
	Coordinator role	Project manager								
		Business analyst								
		Technical designer								
		Tester								
		Unmanaged dependency								

A *dependency-based view* of the data in Table 49 provides the cell count data shown in Table 50 and Table 51.

Table 50 Rock dependency percentages - secondary dependencies

Secondary dependency	Count	Percentage (n=42)
Knowledge	27	64%
Task	10	24%
Resource	5	12%
TOTAL CM: dependency pairs	42	~100%

Table 51 Rock dependency percentages - primary dependencies

Primary dependency	Count	Percentage (n=42)
Requirement	11	26%
Expertise	9	21%
Activity	8	19%
Task allocation	5	12%
Technical	4	10%
Business process	2	5%
Historical	2	5%
Entity	1	2%
TOTAL CM: dependency pairs	42	~100%

This view indicates that 64% of pairs in Rock were associated with a knowledge dependency. That is, attending to requirement dependencies, expertise dependencies, historical, and task allocation dependencies constituted most of the coordination work in Rock. Of those knowledge pairs, most involved requirements (23%) and expertise (21%) whereas task allocation at 5%, and historical dependencies at 2% involved very few coordination mechanisms.

A *coordination mechanism-based view* provides the cell count data shown in Table 52.

Table 52 Rock coordination mechanism percentages

Secondary coordination mechanism	Count	Percentage (n=42)
Synchronisation activities	15	36%
Synchronisation artefacts	8	20%
Impersonal artefacts	4	10%
Boundary spanning activities	1	2%
Boundary spanning artefacts	2	5%
Availability	1	2%
Proximity	1	2%
Substitutability	4	10%
Coordinator role	6	14%
TOTAL CM: dependency pairs	42	~100%

This view indicates that synchronisation activities and the artefacts produced and used during those activities were the predominant form of coordination mechanism because 56% (36% + 20%) of all pairs were for this purpose. In Rock, the coordinator role addressed multiple dependencies. Substitutability, in the form of specialisation in this project was also an important means of addressing multiple dependencies. Impersonal artefacts were also used, and included the various documents developed to guide development.

The majority of coordination mechanisms, as shown in Table 49, address either a single dependency or two dependencies. The main means of coordination was informal negotiation (addressing five dependencies), role specialisation (four dependencies), and for coordination with external parties a coordinator role was used, although this was not taken by a single person but shared by four people in the project.

In summary, Rock used a strategy made up of the following coordination mechanisms. They are shown in Figure 16 in diagram form:

- Synchronisation activities at the frequencies: per project, daily, and ad hoc
- Synchronisation artefacts created or used during those synchronisation activities. This is shown in Figure 16 with a double headed arrow linking synchronisation activities and artefacts. This indicates that synchronisation activities produce artefacts, and those artefacts are used as the project progresses (e.g. work items displayed on a Kanban wallboard)
- Impersonal artefacts created by specialists and used by the rest of the project team to guide their work
- Availability of team members, which was achieved because all project team members worked on the single project and were available simultaneously.
- Proximity of project team members, which was achieved by locating all of the project team close together in a section of a single large project room
- Substitutability of team members, which took the form of role specialisation because almost there was almost no substitutability of roles on this project
- Organisation structure affected availability, proximity, and substitutability. The management of the organisation chose to co-locate this project team affecting availability and proximity. The role divisions, typical in the wider organisation, were maintained. This is shown in Figure 16 with arrows from the organisation structure to availability, proximity, and substitutability indicating that organisation structure influenced these coordination mechanisms

- Coordinator roles were taken by various team members who interacted with external stakeholders on behalf of the team to get project inputs
- Boundary spanning activities performed as and when needed (ad hoc)
- Boundary spanning artifacts produced during these activities. This is shown in Figure 16 with an arrow from the boundary spanning activities to the boundary spanning artefacts
- Project uncertainty led to multiple coordinator roles in the project, and increased the boundary spanning activities required to interact with external parties. This is shown in Figure 16 with an arrow from project uncertainty to coordinator role, boundary spanning activity, and boundary spanning artifact

4.7 SUMMARY

This chapter has presented individual analyses of the four cases. Each case was analysed to determine its dependencies and its coordination strategy. The findings from these analyses are used in the next chapter, Chapter 5, to build a theory of coordination to explain coordination in co-located agile software development projects.

Figure 16 Rock coordination strategy

5. A THEORY OF COORDINATION IN AN AGILE CONTEXT

This chapter proposes a theory explaining coordination in co-located agile software development projects. This theory is the principal contribution of the thesis, and is constructed from analyses of cases presented in Chapter 4, and existing literature on coordination and software development. The theory is presented according to the elements of theory discussed in the final section of Chapter 3.

This chapter has the following structure:

- The coordination strategy concept
- The coordination effectiveness concept
- Boundaries, associations and states of the theory
- Theoretical propositions
- Summary

In the following sections, a theory of coordination is presented, element by element. First, a coordination strategy concept is defined, followed by the antecedents to this strategy. Second, a coordination effectiveness concept is defined. The relationship between these concepts is then elaborated in a discussion of the associations between the concepts, the boundaries of the theory, and the possible states the theory can take. Finally, details of the relationship between the theoretical concepts are stated in the form of propositions.

5.1 A COORDINATION STRATEGY CONCEPT

The initial conceptual framework guiding this study proposed that an agile software development project has a coordination strategy. A coordination strategy was defined as a group of coordination mechanisms managing dependencies in a situation. These mechanisms form a strategy because they are selected consciously by project stakeholders, rather than occurring by chance. Development of the coordination strategy concept involved first identifying the particular coordination strategy of each case (this analysis was presented in Chapter 4 The Cases), then amalgamating these individual strategies to form a generic coordination strategy concept. The non-agile project Rock is included in the following discussion because it provides a contrasting strategy to that of the three agile projects.

The coordination strategy concept comprises three components: synchronisation, structure, and boundary spanning. The following sub-sections define and discuss each of these components in turn.

5.1.1 SYNCHRONISATION

Synchronisation emerged from the case analyses as a common component in each project's coordination strategy. Synchronisation is a relation that exists when things occur at the same time, or are simultaneous (Allen, 1990, p. 1236). In the projects, synchronisation was achieved with synchronisation activities and synchronisation artefacts produced and used during those activities. A typical example is an 'iteration planning session' when the whole team meets together to plan the iteration. This involves selecting stories to work on, breaking down those stories into tasks during a discussion, which may involve understanding requirements, design details, technical constraints, and task interdependencies. Once the stories are broken down into tasks, then the new tasks are stored waiting to be implemented in code. In this example, the meeting is a synchronising activity, and stories and tasks are synchronising artefacts. These artefacts are produced (e.g. tasks are produced from stories) or consumed (e.g. stories are taken from an existing backlog of stories to be processed) as part of a synchronising activity. The following discussion defines the sub-components of synchronisation, which are synchronisation activities and artefacts, and provides examples of how they occurred in the agile projects.

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). Each frequency is now considered in turn.

A project synchronisation activity occurs once during the project. All agile 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 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 thereafter continuously available for consultation by everyone on the team.

An iteration synchronisation activity occurs 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 also 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?" [DP02]

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 by drawing straws to nominate who would demonstrate the latest software version to the business. The demonstrator's job included preparing a script for the demonstration so it would have a coherent flow. This sharing of responsibility for the demonstrations extended the demonstrator's 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 were 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." [DP02]

A daily synchronisation activity occurred in all agile 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.

At 9:25 am, there was a stand-up meeting attended by all 10 project team 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 9:35am. [C, Field note]

The non-agile project Rock also chose to have a daily stand-up meeting at their Kanban board, once the project team size grew beyond a single business analyst, and Host, middleware, and front-end developers joined the project.

Ad hoc synchronisation activities occurred as and when needed in all projects. 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 estimate their size. The purpose was to maintain the backlog so 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.

Project Rock, the non-agile project, differed from the agile projects because there were no iterations. A daily meeting at the Kanban wall board, and ad hoc meetings were the chosen mechanisms for synchronising in that project team.

Synchronisation artefacts are physical things generated and used during synchronisation activities. These artefacts contain information used by team members in accomplishing their work. These artefacts store project information such as design decisions and may serve as objects to focus discussion. The nature of an artefact can be publicly visible to the whole team at a glance (e.g. on a whiteboard in the team

workroom) or largely invisible but available (e.g. located in a shared computer file). An artefact can be physical or virtual, temporary or permanent.

Synchronisation artefacts include working software, product backlog, epics, stories, tasks, the wallboard, Done lists, burn-down charts, project information repositories such as wikis or the Rally™ application (a software product for supporting the Scrum methodology), and shared document systems. In Land, there was no wallboard; stories were developed and stored in a shared document, and a design specification document was the primary source of project design information. This document evolved because it was adjusted and added to at each iteration planning meeting as design details were elicited from the customer representatives, and discussed and finalised in whole team discussions. The single developer on the project described this document as the “*one source of truth*” [BT01] in the project.

In summary, the coordination strategy in the agile projects involved multiple coordination mechanisms for synchronising the project team. These mechanisms were either activities or artefacts. Activities occurred at different frequencies: per project, per iteration, daily, and ad hoc, and artefacts were produced and consumed during these activities.

Figure 17 illustrates the synchronisation component of a coordination strategy, and its sub-components.

Figure 17 Synchronisation sub-components

Project Rock had an additional coordination mechanism coined ‘impersonal artefact’. In organisation studies coordination literature, reviewed in the literature review in Chapter 2, impersonal coordination is achieved with the use of policies and procedures. Based on this, an impersonal artefact is defined as an artefact produced to enable

coordination within the team and project boundaries. Such an artefact is produced in its entirety once and typically near the start of the project. The artefact can be physical or virtual, but is permanent and available to the whole team. This artefact may undergo unplanned changes during the project, although this is considered undesirable; these artefacts are not initially conceived as documents that can be revisited and revised as the project progresses. The traditional project documentation produced in Rock (i.e. use case specification, system design specification, unit test document, and defect documentation) fits this definition. Although Land produced a document called a system specification, their system specification was an evolving document. Since impersonal artefacts occurred only in the non-agile project, this coordination mechanism is not included in the final generic coordination strategy concept.

5.1.2 STRUCTURE

Structure is another component of coordination strategy found in the case analyses. Structure is used in its common sense as the arrangement of, and relations between, the parts of something complex. Three categories of coordination mechanism have structural qualities: proximity, availability, and substitutability. The following discussion defines each of these categories.

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 & 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 Chapter 4 The Cases. Rock also had a project team in close proximity because this was the preferred project arrangement in that organisation where most development projects were performed using Scrum. Rock was located on the same floor as these Scrum projects, so this single non-agile team were also in close proximity, and this was commented on positively:

"Because you didn't have to – one, it was quick. And again, the face to face always helps. We were able to explain right away. And the turn-around was quick. They would say right away that they would work on it and what the issue was if they found out after. Same with myself. The tester was beside me as well. So you had the test analyst, you had the three components people,... right beside each other." [GT01]

Availability is achieved when team members are continually present and able to respond to requests for assistance or information, or to participate in project activities when needed. Availability was readily achieved 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. The tester in Storm noted:

"But you know the developers are right there and we just turn around and involve one of them in the discussions. " [CT04]

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." [BP01]

Rock also had a project team whose members were available to one another. Although some team members worked from home, this was on scheduled days and the team used SMS chat to discuss issues with those present in the project room, and those at home. Rock participants did not report this as a constraint on the project.

Substitutability is a structural element achieved when a team member has 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. This skill redundancy, or substitutability, improved workflow. For example, when a Storm developer saw a test task 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:

"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. " [CT04]

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, she explained:

"... 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." [DP01B]

In Rock, the non-agile project, substitutability was not encouraged and high levels of role specialisation were apparent. Each team member had a specialised role and crossover of roles was not considered a possibility. Therefore substitutability in Rock was not reported, specialisation was the coordination mechanism used in its place. That is, when there was too much work for one specialist to complete in a timely manner, additional project team members with the required skill set were seconded from the wider organisation, or new staff were employed to maintain project progress.

Figure 18 Structural sub-components

In summary, a coordination strategy includes a structure component, which consists of coordination mechanisms involving proximity, availability, and substitutability. Each contributes to project structure by involving the organisation of people and their roles. Figure 18 illustrates structure and its sub-components.

5.1.3 BOUNDARY SPANNING

Boundary spanning was another component of coordination strategy emerging from the case analyses. Boundary spanning is the act of linking two or more groups of people separated by location, hierarchy, or function (Levina & Vaast, 2005). Boundary spanning is important for coordination because coordination mechanisms for boundary spanning address constraints on project progress due to the action or inaction of external parties. In this study, boundary spanning occurred when someone within the project interacted with other organisations, or other business units, not directly involved in the project to achieve project goals. A notable difference between the synchronisation and structure components of coordination, and boundary spanning, was that boundary spanning is not normally team-wide, nor is it performed in subgroups. Individuals on the team perform boundary spanning when the project

needs information, support, or resources from other organisational units or external organisations.

Three categories of boundary spanning coordination mechanisms were identified in the projects: boundary spanning activities, production of boundary spanning artefacts, and coordinator roles. Each is defined as follows.

A boundary spanning activity is performed to enable reciprocal interaction between project team members and other parties external to the project, to meet project goals. In the projects, boundary spanning activities occurred at three frequencies: per project, per iteration, and ad hoc.

Unlike the situation with synchronisation, a daily boundary spanning activity was not included as a category of boundary spanning activity. Even though a daily boundary spanning activity was identified in project Storm. In Storm, this coordination mechanism was employed because of a unique circumstance when the end-user group (engineers) had 'free time' at a regular time each day for creative thinking and problem solving. The project team members took advantage of this period to consult with the engineers on requirements, design issues, or other project-related work. This is an unusual situation between end-users and a project team, and for this reason 'daily boundary spanning' is not included in the generic coordination strategy concept.

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 the engineers involved numerous 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 per iteration when the project 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 used in Storm was to transfer a selected project team member to work within another business unit for an extended period. This team member could then return to the team with knowledge about requirements or technical aspects of another system that the new system had to integrate with. The situation was different in Silver, where boundary spanning between the team and customer proxies was achieved by adhering to a strict per-iteration schedule to ensure the team had regular contact and feedback from their customer. Ad hoc boundary spanning activity in this project was severely limited due to problems with customer availability. Rock contrasted with the three agile projects in its boundary spanning activities because, in that project, all boundary spanning activities were ad hoc.

Boundary spanning artefacts are physical things produced to support boundary-spanning 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. For example, Land produced 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." [CT04]

Coordinator role is another coordination mechanism to accomplish boundary spanning. A coordinator role occurs on the project team when someone initiates and sustains reciprocal interaction with other external parties to the project to meet project goals. This person acts as a primary point of contact between project team members and other organisational units or external parties. 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." [CT04].

Hoda, Noble, and Marshall (2010) found a similar coordinator role in their research on agile teams. This was reported in the literature review in Chapter 2 in the section on coordination in agile software development.

In Rock, multiple coordinator roles were observed. This was influenced by the strict role specialisation on that project. Each specialisation would include one senior person

who took the role of coordinator with a different group of stakeholders. For example, the Host technical lead would coordinate with the Host business unit, the business analyst would coordinate with the external organisations, and the tester would coordinate with the testing business unit.

In summary, a coordination strategy includes coordination mechanisms for boundary spanning. Figure 19 illustrates boundary spanning and its sub-components. Boundary spanning mechanisms are of three kinds: boundary spanning activities, boundary spanning artefacts produced to support boundary spanning, and a coordinator role.

Figure 19 Boundary spanning sub-component

The preceding discussion shows that, in the three agile projects, multiple coordination mechanisms were used to achieve coordination. Together they form a coordination strategy consisting of three distinct categories of coordination mechanism. There were coordination mechanisms for synchronising the whole team – bringing them together at regular and irregular intervals. There were artefacts produced and consumed as part of these synchronisation activities that acted as coordination mechanisms. There were coordination mechanisms for structuring the team, so they were close together, available to one another, and could perform each other's tasks when necessary to avoid interruptions to project progress. Finally, there were coordination mechanisms for boundary spanning. These mechanisms generally do not involve all of the project team, only specific individuals, but they are necessary to ensure a steady flow of requirements, information, support, and resources to maintain project progress. Boundary spanning artefacts may be produced to support these activities. A coordinator role is another coordination mechanism for boundary spanning. This role maintains needed links with external parties, freeing the remainder of the project team

to focus on their project work. The complete coordination strategy concept, which includes each of these coordination components is illustrated in Figure 20.

Figure 20 The coordination strategy concept

5.2 COORDINATION STRATEGY ANTECEDENTS

Project complexity, project uncertainty, and organisation structure were found to influence the coordination strategy of the projects. Project complexity and project uncertainty were proposed as influencing factors in the initial conceptual framework guiding the study, whereas organisation structure emerged during data analysis.

5.2.1 PROJECT COMPLEXITY

Project complexity led to changes in a project's coordination strategy by influencing coordination mechanisms for synchronisation and for boundary spanning. For example, in Storm, coordination mechanisms for synchronisation were adjusted when the story breakdown meetings were unmanageably complex and very long, because they involved 10 project team members. To change this, the team decided to shorten the iteration from 2-week to 1-week. This reduced the number of stories worked on in the iteration, and reduced the length of time it took to break down those stories into tasks. This is evidence that project complexity influences synchronisation coordination mechanisms, both synchronisation activities, and production and use of artefacts.

Project complexity also led to a change in boundary spanning coordination. A coordinator role was introduced into those projects that required inputs from multiple external parties. This happened when many different systems needed to be integrated with the new system, which necessitated extensive negotiation with external parties. For example, in Storm there were two coordinator roles and in Rock there were four. These roles were necessary when the project team needed information about the format of input data from external systems, and when and how it would arrive into the new system. When this happened, some team members would act as coordinators with various external groups both internal and external to the organisation to gain this understanding and communicate it back to the project team. This evidence shows that project complexity led to the need for boundary spanning coordinator roles on the project teams, both agile and non-agile.

5.2.2 PROJECT UNCERTAINTY

Project uncertainty influenced synchronisation coordination mechanisms. In Silver, when requirements were uncertain and the project team could not get timely feedback from their customer about requirement details or priorities, they maintained their iteration duration, but switched the stories they were working on to maintain project progress. A change to the wallboard (a synchronisation artefact) was introduced to

assist with this problem; the wallboard was changed to display blocked stories and tasks. This acted as a visible reminder to the team about halted work that might need to be completed at a later date. This is evidence that project uncertainty influences synchronisation activities and associated synchronisation artefacts.

It is interesting to note that there were uncertain situations during one project that did not lead to any apparent change in coordination strategy. In Storm, the technical environment was selected at the start of the project and did not change during the project. At one point, this caused a problem for the project:

“External change, so we have quite a big one where my team literally started coming to me and saying “what shall I do today?” and it was caused because the internal teams were having a debate about the technology set that they were using and that was going to affect what we used. So suddenly, our technology choice that we were building had potential to flip, and so we didn’t want to push forward in something that wasn’t a certainty. “Oh well, that might get changed on us in two weeks time” so we kind of had this weird sort of hiatus.” [CP01]

This indicates that uncertainty can impact projects, and there may be no coordination strategy for managing it.

5.2.3 ORGANISATION STRUCTURE

Organisation structure is the way the functional units of an organization are compartmentalised to meet organizational goals (Galbraith, 1974). In Land, the analysis presented in Chapter 4 identified a matrix organisation structure as detrimental to the project. This matrix structure reduced the proximity and availability of project team members to one another. In Land, the project team members were neither all co-located, nor working on a single project, and this was perceived by the project leader as a major issue in the project.

The other projects, Silver, Storm, and Rock, all had project-based organisational structures whereby the project was organised primarily with full-time, co-located project team members working only on a single designated project. In these projects, availability and proximity were seldom mentioned, although in Rock, project team members commented that co-location and constant availability were beneficial. This is evidence that organisation structure has an effect on proximity and availability.

These three antecedent factors influencing the coordination strategy of the three agile projects are shown in Figure 21.

Figure 21 Coordination strategy antecedents

5.3 A COORDINATION EFFECTIVENESS CONCEPT

The initial conceptual framework presented in Chapter 2 Literature Review, proposed coordination effectiveness as a concept associated with a coordination strategy. A

complete characterisation of coordination effectiveness emerged from analyses of the three agile cases, and the coordination literature. This characterisation is presented in this section. How coordination effectiveness was viewed in case Rock, the non-agile case, is discussed at the end this section.

To develop a concept of coordination effectiveness all participants were asked to respond to these questions at the end of their interview session:

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?

Two components of coordination effectiveness emerged from the case data, one implicit, and the other explicit. The literature broadly defines explicit coordination as that which occurs when two or more team members use overt mechanisms such as schedules, plans, and procedures, and send communication messages to one another using formal or informal, oral or written, transactions to integrate their work (Espinosa, et al., 2004; Nonaka, 1994; Rico, et al., 2008; W. P. Wang, et al., 2001). Implicit coordination occurs when team members anticipate the actions and needs of their colleagues and adjust their behaviour accordingly without preplanning or direct communication (Nonaka, 1994; Rico, et al., 2008; W. P. Wang, et al., 2001).

5.3.1 EXPLICIT COORDINATION EFFECTIVENESS

Research participants identified three explicit components of coordination effectiveness: *right thing*, *right place*, and *right time*. These components, although conceptualised as three different factors, were sometimes present together in the transcript quotes. For example, (this chapter shows sections of interest within quoted blocks of text in bold):

*"You know there was a decision to be made, but they couldn't make it without knowing the technical implications. So they wanted to wait for me to get back. So that probably interfered with coordination, **not having me present all the time.**" [BT01]*

This response, from a Land participant, shows the three factors intermingled. Right thing (*me*), right place (*present*), and right time (*all the time*). Other responses also noted the importance of accessibility and time, for example:

*"As long as these **people were accessible** to Mark, the communication would be flowing, but if these people were not accessible, or **there was a time lag**, then coordination becomes problematic." [BT01]*

Time was a concern in Storm and perceived to impact coordination:

*"You want **someone** to be able to react **at the time something happens**." [CP01]*

Right time, right place, and right thing, were also identified by a participant in Silver. In the following quote, the thing (i.e., the software), needed to be in the right place at the right time – that is, immediately integrated.

*"It [user stories] allowed a cohesive flow of the development so we didn't end up developing **something** that couldn't be **immediately integrated into the project**." [DT03]*

Prior coordination research also proposes right thing, right time, and right place as factors important in coordination. These factors are identified in an article describing a Process Handbook (reviewed in Chapter 2 Literature Review). In describing this Handbook, Malone et al. (1999) conceptualise these three properties as elementary flow dependencies. The grounded evidence from this study finds these properties to be outcomes of a coordination strategy. That is, when dependencies are well-managed with coordination mechanisms, then the right things will be in the right place, at the right time.

5.3.2 IMPLICIT COORDINATION EFFECTIVENESS

Implicit coordination effectiveness is the second component of coordination effectiveness found in the agile cases. Four implicit components were identified and named as follows:

1. *Know why*
2. *Know what is going on and when*
3. *Know what to do and when*
4. *Know who is doing what*

The *know why* or *shared goal*, component is about each individual working on the project understanding the overall project goal and understanding how a task contributes to that overall goal. The importance of a shared goal emerged in each project:

*"To me, coordination is basically running in the same direction, meaning **there is a shared goal** and people know what they are supposed to do." [DP02]*

*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.” [BT01].*

In other instances, shared goal emerged when participants reflected on the lack of shared goals, and how this was detrimental to their project:

*“No one is able to say ‘this one is more important than that one, so stop working on that one’. There are different groups of people all saying ‘this is the top priority’. And no one able to say, ‘Ok, you have got three things of equal priority; in fact they are in this order of 1, 2, and 3. No one able to make that decision, or get agreement on those relative priorities, and so **you are left in a position of three equally important high priority things...**” [BP01]...“Which was another major issue for the project, that they all had other really equally important, if not more important stuff going on...” [BP01]*

Another perspective on shared goal is the when a project team member was aware of the shared goal, and therefore knew when they were not contributing towards it, and the impact this had on the project:

*“...and we saw that with this project, when we got comments like, ‘did not want to let the team down’ that was an **understanding of the impact that not delivering on something** was going to do to the rest of team and therefore on our ability to get that end game of being seen to be successful.” [BP01]*

A participant from Storm, compared their agile project with previous projects he had worked on:

*“...just being in the situation where you have a good understanding... you know enough that you know what you are a part of, rather than being told to work on a little corner of something and **never knowing what you are actually doing it for.**” [CT01]*

This finding is supported by agile software development research conducted by Rising and Janoff (2000). They studied three teams, and said of Scrum:

For those who know rugby, the image is clear. Teams work as tight, integrated units with each team member playing a well-defined role and the whole team focusing on a single goal. In development teams, each team member must understand his or her role and the tasks for each increment. The entire team must have a single focus. (Rising & Janoff, 2000, p. 30)

This evidence supports the inclusion of *know why* or shared goal as a component of implicit coordination. *Know why* is when team members have a shared understanding of the project goal and project priorities.

The *know what is going on and when* component is about 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.

*“...because I think the result of that was ... decisions took longer than they should have. So when they were talking about, ‘Ok ...what is this part of the system going to look like’, well **it depends on what those guys of doing, and what these [other] guys are doing.**” [BT01]*

*“So those things [sprints, stand-ups, retrospectives, planning sessions] were the bread and butter that meant **we knew the things that we were doing.**” [DT03]*

Rock, also used a wallboard, although it was a Kanban board, which has a somewhat different function to a Scrum wallboard (described in Appendix L Rock Case Description). The Rock team also had a daily stand-up meeting in front of their board. They found this contributed to the project team member’s knowledge about what was going on in the project, and when it was likely to occur:

*“... [team members] don’t get a surprise [when someone says], ‘here, something’s ready for you to test now’. I wasn’t expecting that for three weeks. So in that way the progression of the work items across the wall did help because **you knew that something was coming up and you could see what it is.**” [GT02]*

This evidence supports the inclusion of ‘know what is going on and when’ as a component of implicit coordination.

The *know what to do and when* component is about 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. The following examples show that project team members perceived being well-coordinated involved knowing what to do and when to do it:

*“Everyone knows **what is expected of them and when** and what the impact is if they don’t do that, how that is going to impact on other people.” [BP01]*

*“My team, I can tell if they are well coordinated by the fact that they are not confused about what they have to do each day I guess, **if they know what they are doing and what they have to do next**; then they are well co-ordinated.” [CP01]*

*“Because **I know what to do**. Each time I don’t get lost, and I, even having a lack of information because of my English condition, **I can know what I have to do**, what people expect me to do.” [CT01]*

*... just being in the situation where **you have a good understanding of everything that you need to do.**” [CT02]*

A developer from Land, compared their agile project with previous projects he had worked on:

“Or an agreed upon process. I have definitely worked in smaller companies, where no one has actually gone to the trouble of spelling out the process, it has just been assumed...you know,...‘how are we going to capture requirements’, ‘what are we considering “done”’. You know...a lot of that stuff, in a lot of organisations, a process is not really clearly defined. No one

has gone to the trouble of saying...when you have done this, then we can say that this is done, and it will go to the next stage.” [BT01]

This evidence supports the inclusion of *know what to do and when* as a component of implicit coordination.

The *know who is doing what* component is about each individual on the project knowing what tasks others are currently working on. Examples include:

*“I guess from a developer point of view, just being in the situation where you have a good understanding of everything that you need to do. **Everything that is happening; everything that is happening around it.**” [CT01]*

*“...people need to find out **who is doing what** in a flexible way, and in a way that they decide for themselves, so they own this process” [DP02]*

This evidence supports the inclusion of *know who is doing what* as a component of implicit coordination. This is when everyone on the team knows who is doing what and how their work fits with other peoples work.

Responses from participants focused more on implicit factors, than explicit factors. This could be a reflection of the well-resourced nature of the projects – there were few issues with progress being affected by a lack of needed hardware, software, licenses, or additional skilled people in the agile projects. This, in turn, could be related to the project selection criteria (described in Chapter 3 Research Methodology). Since all projects had to be of some importance to the organisation where they occurred, this meant the resources necessary to carry out the project were readily available.

At this point, an additional component of implicit coordination effectiveness is introduced into the coordination effectiveness concept. This component comes from the literature on software project team coordination. Faraj and Sproull’s (2000) research on expertise coordination in software teams identified three components of expertise coordination (discussed in Chapter 2 Literature Review, in the section Coordination in Teamwork Studies on page 68). The first two, ‘knowing where expertise is located’ and ‘knowing where expertise is needed’, are implicit aspects of coordination. The third, ‘bringing needed expertise to bear’, is an explicit aspect of coordination, which is an example of the *right thing* coordination component. Although there was no direct evidence for expertise coordination in the findings from the agile projects, Faraj and Sproull’s arguments and findings are compelling. Therefore, *Know who knows what*, is included in the characterisation of implicit coordination effectiveness.

The preceding analysis leads to the following definition of coordination effectiveness:

Coordination effectiveness occurs when the entire agile software development team has a 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 individuals 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.

Figure 22 illustrates this definition.

Figure 22 The coordination effectiveness concept

5.3.3 COORDINATION EFFECTIVENESS IN PROJECT ROCK

Rock was a non-agile project, and an analysis of project team member's perceptions of project coordination effectiveness focused on similarities or differences between this project and the agile projects.

Rock used a hybrid development methodology, which included two practices common to agile software development projects, a daily stand-up meeting in front of wallboard (a Kanban wallboard), and continuous integration of the software. In all other respects, the project was organised in a very different way to the other agile projects. A Kanban

wall is very different to a Scrum wallboard. The Kanban board in Rock was organised by project team member role, and these roles were heavily dependent on the existing system architecture in the organisation.

Perceptions of coordination effectiveness in Rock were mainly the same as those emerging from the agile projects. Where Rock differed from the agile cases was in the participant perceptions of how coordination effectiveness is achieved, that is, what coordination mechanisms are most appropriate for achieving coordination effectiveness. Team member responses in Rock were influenced by the distinct roles taken within the team: Host developer, front-end developer, business analyst, and tester. Participants expected their roles would determine who does what, who knows what, and that someone in a particular role, the business analyst in this case, would ensure that work was progressing.

In Rock, explicit coordination effectiveness was perceived as a smooth flow of work from one group, such as developers, to another group, such as testers, as this excerpt shows:

*"Or **somebody's not ready for something** and then all of a sudden they're holding it up because they weren't expecting something to arrive so soon. So it's both ways, and if somebody all of a sudden gets flooded with work, then they can get help with that as well because that often happens with testing. You are sitting there, you do your preparation and then there's weeks go by and the developers are all working away and then all of a sudden everybody delivers their code and 'okay the code's all finished, why can't we have it?'"*
[GT02]

This is evidence for explicit coordination effectiveness because it describes how a code module (the right thing), is transferred from the developer to the tester (right place), at the expected time (right time) thus avoiding bottlenecks whereby some people have too much work to do while others wait for their outputs.

Implicit coordination effectiveness in Rock also emerged. Participants identified a shared goal, *know why*, as contributing to coordination effectiveness:

*"The team dynamic. **The core team members all understood, once we'd got into the project, what was needed.** The people doing all the coding were **all on the same page** and all competent people with good skills."*
[GT02]

When the team size grew quickly from seven to 15, the shared goal began to break down:

"Well, that's another thing that we probably could have done better, ... had a bit more of an overview of this is what we're trying to do. But they are thrust in and [told] "do this" with no context maybe. So, because we're just trying to get the work done, I know for some of the other Host developers that came

*on, it would have probably been really good for me to sit down with them and talk about the project and give them a bit more context about **what they're actually doing, and why.***" [GP01]

Evidence for *know what is going on and when*, was also found. This was influenced by the presence of a wallboard to display work items, as follows:

*"... is that people expect things when approximately – they don't get a surprise [when someone says]' here something's ready for you to test now'. I wasn't expecting that for three weeks. So in that way the progression of the work items across the wall did help because **you knew that something was coming up and you could see what it is.** You could see from afar that there was something, and you would go and have a look, and see what it was that somebody had moved."* [GT02]

Knowing what to do and when, was identified as contributing to project coordination effectiveness in both negative and positive comments:

*"...you need somebody to be making sure that things are going as they are supposed to without anybody **sitting on something just because they don't know what to do.***" [GT02]

*"And being good at what they do and being able **to understand what was required and how it fitted in.***" [GT02]

In Rock, the strict role divisions in place heavily influenced perceptions of who is doing what. A team member would assume they knew what another person on the project was doing because they understood that person's role and what that role typically involved (i.e. Host developer, tester, middleware developer). One developer reflected on this when he said, *"Their individual role in relation to the other roles, because if that's defined at the start then the coordination will just flow."* [GT01]

In Rock, evidence to support the component, *know who knows what*, was found, whereas no evidence for this was apparent in the three agile projects. After considering Faraj and Sproull's (2000) research on expertise coordination in software teams (as described above) this component was added to the model of coordination effectiveness. The strict role divisions in Rock could explain why evidence for *know who knows what* appeared in the non-agile project and not the agile projects. In Rock, each person had different expertise, therefore knowing who knows what, was important. In the agile teams without such strict role divisions, knowing who knows what might be less important because role divisions are less strict. Substitutability of roles is valued and encouraged. This excerpt from the interview with a Rock project team member indicates how roles defined not only who knows what, but also who does what:

*"...for example, when faced with an issue, who addresses the issue? And acceptance of the result. For example, [if] **you know that it is something***

that we need to sort out on the Host [system], then you accept that we [the host developers] will sort it out.” [GT01]

Based on these arguments and the evidence, *know who knows what*, has a tentative place in the final theoretical coordination effectiveness concept.

Evidence from the non-agile project and the three agile projects shows a similar pattern of implicit and explicit components. Across all projects, although their coordination strategies were different, their perceptions of coordination effectiveness were similar. This indicates that the concept of coordination effectiveness in its current form may apply to both agile and non-agile projects.

At this stage, this chapter has presented evidence for a coordination strategy concept, antecedents influencing that coordination strategy, and a coordination effectiveness concept. In the following sections the relationship between the two principal concepts – coordination strategy and coordination effectiveness is discussed.

5.4 BOUNDARIES, ASSOCIATIONS, AND STATES

The elements of theory include not only theoretical concepts, but also associations between those concepts, various states of the theoretical system, and boundaries. This section discusses the boundaries within which the theory of coordination in agile software development projects is expected hold, discusses the associations between the two principle theoretical concepts, and then proposes certain system states.

The most salient boundary of this theoretical system is the co-located agile software development project. This was determined when the conceptual framework was developed, and was narrowed down during the research design phase when theoretical and practical constraints determined case selection. After case analysis, the boundaries within which the theory applies can be determined more precisely still, because they are informed by the profile of the projects in the cases studied. These boundary conditions are:

1. A software development project using practices from Scrum, or Scrum and Extreme Programming. Exactly which practices are adopted is not specified except that the project must use iterations (sprints) of one or two weeks. This is the iteration duration used in the projects in the cases studied.
2. A distinct and identifiable project team of 2 to 10 people who work concurrently and full-time on the project, and who are located in close proximity within the same room in direct line of sight of one another. This team

may have customers as part of the team, or the customer may be external to the project team.

3. A project with a clear business purpose that is either providing a software product for another business unit within the organization, or for an external organisation.
4. A project with a distinguishable customer or proxy customer. This can be a single person, a group, or groups of people.

There is one principal association proposed for this theory of coordination - an agile coordination strategy comprising synchronisation, structure, and boundary spanning components, is associated with a high level coordination effectiveness engaging both implicit and explicit components.

This theory builds on evidence from three distinct cases and certainty about this boundary can only be achieved if a larger number of independent cases are analysed to contribute to the theory, or if the theory is tested using quantitative research methods and statistical sampling. As it stands, the theory is only relevant in describing cases identical to or very similar to those described in the case descriptions (the cases are described in detail in Appendices H, I, J, K). Their commonalities informed the boundary items above.

Systems states are another element of theory. Evidence for two distinct states for the coordination strategy concept is apparent in the cases. No distinct states are identifiable for the coordination effectiveness concept, although further research into the interplay between implicit and explicit coordination effectiveness could uncover such states.

Two states are apparent among the projects' coordination mechanisms. The first depends on whether the customer is considered as part of the project team or is an external party to the project. When the customer is part of the team, boundary-spanning activities are considerably reduced. Evidence for this came from Land data. In Land, the customer was part of the team. Three people from the wider organisation were selected as proxy customers (the true customer was the public), and were treated as part of the team. They participated in all of the synchronisation activities such as the weekly sessions for discussing requirements and design details, and gave feedback on the current state of the working software. Because these people formed part of the team, there was little need for the project team members to have separate regular or ad hoc boundary spanning activities with these people. In Land, no boundary spanning

activities were reported. Issues with contacting team members did occur, but that was because they were not co-located.

In Storm, Silver, and Rock customers were external parties to the project, and not considered part of the team. In these cases, extensive boundary spanning activities took place. Furthermore, Storm and Rock, both highly complex projects involving multiple external stakeholders, and systems requiring integration with the new system, initiated coordinator roles to manage the additional coordination required to negotiate with these external parties. In addition, on these two complex projects, domain specialists were brought into the project team, to increase access to requirements and improve the teams understanding about the existing systems within the organisation. This increased the synchronisation activity coordination mechanisms and reduced boundary spanning coordination mechanisms, because there were more ad hoc consultations within the project team.

The second state is when impersonal artefacts, may substitute for iteration level synchronisation activities, and their associated synchronisation artefacts. This was identified by comparing Rock, which had four impersonal artefacts (i.e. use case specification, system design specification, unit test document, and defect documentation) and no iterations, and the other projects with no impersonal artefacts and synchronisation activities at each iteration. For example, in Silver the project team had four different activities every two weeks that brought the whole team together to focus on sharing information about requirements, design, and other project issues. In Rock, this coordination was primarily achieved when individuals consulted one of their project documents.

These states are summarised as follows:

1. Customer on team - boundary spanning
 - a. Customer as part of project team requires a coordination strategy including coordination mechanisms for synchronisation and structure to achieve coordination effectiveness.
 - b. Customer external to project team requires a coordination strategy including coordination mechanisms for synchronisation, structure, *and* boundary spanning to achieve coordination effectiveness.
2. Impersonal artefact – iteration synchronisation
 - a. A coordination strategy using iteration synchronisation does not require impersonal artefacts to achieve coordination effectiveness.

- b. A coordination strategy *without* iteration synchronisation will require impersonal artefacts to achieve coordination effectiveness.

Note that the Rock coordination strategy does not form part of this theory because that project was non-agile. Rock differed from the agile projects in its coordination strategy. In Rock, impersonal artefacts were part of their coordination strategy, which means that although Item 2 is interesting this state is not relevant for a theory explaining coordination in *agile* software development projects. It is only useful when comparing agile and non-agile projects. Furthermore, in Rock there were no project or iteration-level synchronisation activities. Rock also differed in that the project team members' roles were not highly substitutable, unlike in the agile project teams. Coordination effectiveness, however, was perceived to be similar in all cases – agile or non-agile.

In summary, Figure 23 illustrates the complete theory of coordination in co-located agile software development projects. The numbers shown on the diagram indicate theoretical propositions linking the two concepts, and these propositions are presented in the next section.

Figure 23 Coordination in co-located agile software development projects

5.5 THEORETICAL PROPOSITIONS

Theoretical propositions are the final element of theory addressed in this thesis (Dubin, 1978). They define the association and direction of the relationships between the concepts in a theory. The proposed coordination theory has nine propositions. They are

illustrated in Figure 23 (in the previous section), and are defined in the following discussion.

The primary relationship proposed in this theory is that the coordination strategy employed in an agile software project leads to coordination effectiveness. 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 reduces the need for 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.

Evidence for this emerged in the Land data, where the customer proxy was part of the team and participated in Sprint activities.

*"We had a **weekly catch up** [that] 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." [BP01]*

This excerpt is evidence for a coordination mechanism (i.e. "weekly catch up") influencing project coordination effectiveness (i.e. "everyone was on the same page", indicating the project team had a shared goal).

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

Proposition 1b. *When the customer is an external party to the project, a coordination strategy that includes synchronisation, structure, and boundary spanning coordination mechanisms improves project coordination effectiveness. 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, making no distinction between the effect of coordination strategy on implicit and explicit components of coordination effectiveness. Many coordination mechanisms contribute to both implicit and explicit coordination. Some coordination mechanisms tend to promote implicit coordination, whereas others promote explicit coordination. The following propositions reflect this tendency. Note, in the following discussion the components of implicit coordination (defined in the previous section defining coordination effectiveness), are italicised so they are easy for the reader to identify.

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*”[DT01], or by attending and participating in the daily stand-up meeting, or other regular or ad hoc meetings. Evidence from project Storm shows this relationship:

*“...everything that we know about what we do, what we are building is through those meetings. The idea has always been that if you are doing something ..., you have to decide if it is important enough, but if it is important enough that other people need to know it, and they might disagree with you, then **you should grab everyone around the table and have that kind of talk.** [CT01]*

This excerpt is evidence for a synchronisation activity (i.e. informal face-to-face negotiation) at ad-hoc frequency (these meetings were held as and when needed) influencing implicit coordination effectiveness, in particular, *know why* or shared goal (i.e. “*everything we know about what we do*”).

A second, and more complex, example from Storm involved three different coordination mechanisms all influencing implicit coordination effectiveness. This excerpt is called Excerpt A, and is used in developing arguments to support propositions 2, 3, and 4.

*“We’ve got the [Scrum wallboard], but you wouldn’t, we don’t do locks, so you can get people working in the same code area, but generally..., **people are ... aware of what other people are doing**, sorry, **in the same room**, people are kind of aware of **what people are working on**.” [CT04]*

So this awareness, where does this awareness come from? [Researcher]

*“The **board and cross-team talk**. But then, even if you do edit the same files, you can merge them again. Or if there was significant change, if someone declares,...that ‘I’m [going to] rewrite this whole thing, it needs complete refactoring’ or ‘it is just so ugly the way we were doing it’ they will **announce that to the team**. They’ll say, ‘no-one bother going into this module ‘cos I’m just going to rip it to bits’. ” [CT04]*

Excerpt A provides evidence that a coordination mechanism categorised as an ad hoc synchronisation activity (i.e. “*cross-team talk*”) contributes to implicit coordination effectiveness, in particular *know who is doing what* (i.e. “*people are aware of what other people are doing*”). Furthermore, this excerpt is also evidence that a coordination mechanism categorised as a synchronisation artefact (i.e. “*Scrum wallboard*”) influences implicit coordination effectiveness, in particular *know who is doing what*.

Based on evidence of this nature, the following is proposed:

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*.

For example, Excerpt A shows that a coordination mechanism categorised as proximity (i.e. “*in the same room*”) influences implicit coordination effectiveness, in particular *know who is doing what* (i.e. “*what people are working on*”).

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. For example:

“But if there is a sticky problem [that] is right inside our team, if you are blocked, need some help, to tap anyone else on the shoulder and grab them and get them to help unblock you...” [CP01]

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. For example, in Storm with everyone similarly skilled in Java, the developer said:

*“...we all know, since we are in a team of four or five people, **what each person is going to be best tackling**. And you just go ‘yes’, I will do that, and no one argues. We just know which ones to grab next. That is pretty much it. We just grab the next bit of work off the board and go for it.” [CT01]*

This leads to the following:

Proposition 3. *Structural coordination mechanisms including 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. For example:

*“...they [the engineers] literally sit in the room next door, so **we jump up from our desk and go over to their desk** and talk with them or bring them into our room, get them to sit down and look at it over our shoulder and talk through it with us. So it is good collaboration level, it is just random who we get, or a little bit random who we get, and it has been narrowing down for [Startpoint] project just the same way it narrowed down for the [Carboy] Project. Each one tends to narrow down to certain people you find who can **tell you the right information** and provide that customer relationship.” [CP01]*

This example shows an ad hoc boundary spanning activity (i.e. jumping up from your desk and going to the next room to consult with an end-user) increasing explicit coordination effectiveness (i.e. accessibility to the right information). Therefore:

Proposition 4. *Boundary spanning coordination mechanisms including boundary spanning activities at all frequencies – project, iteration, and ad hoc, their associated boundary spanning artefacts, and a coordinator role increase 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. For example:

“We started with very, very structured two week iterations. What we found was, when we then took all of the stories from the backlog 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 with 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.” [CP01]

Therefore:

Proposition 5. *To reduce project complexity increase the frequency of iteration and ad hoc synchronisation activities. 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 project uncertainty, to maintain synchronisation activity frequency and production of associated artefacts, change the priority of stories.*

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 of high project complexity when 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, this can increase boundary spanning coordination mechanisms. For example, in Storm, a highly complex project, the tester on the team took on a coordinator role between the engineers and the project development team:

*"They [the engineers] always pop down and talk to me [the tester] 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."*
[CT04]

Therefore:

Proposition 9. *To reduce project complexity when the customer is not part of the team, introduce a coordinator role.*

These propositions, along with the preceding definitions of concepts, boundaries, associations, and system states form a theory of coordination explaining how coordination mechanisms in agile software development projects work together to influence the coordination effectiveness of such projects.

5.6 SUMMARY

This chapter has presented the central contribution of the thesis, which is a proposed theory of coordination in co-located agile software development projects. The theory is based on evidence from within-case analysis presented in Chapter 4, and further cross-case analysis presented in this chapter. There are two principal theoretical concepts in the theory: coordination strategy and coordination effectiveness. The coordination

strategy concept has the components synchronisation, structure, and boundary spanning. A coordination strategy is influenced by project complexity, project uncertainty, and organisation structure. The coordination effectiveness concept has implicit and explicit components. The relationships between the two principle concepts were presented along with the theoretical boundaries and states of the theory. Finally, nine theoretical propositions were presented explaining the direction of the relationships between each of the concepts and components of the theory.

The final chapter, Chapter 6, answers the research questions posed in Chapter 1 and elaborated in Chapter 2, and concludes the thesis.

6. DISCUSSION AND CONCLUSION

This chapter answers the research questions first posed in Chapter 1. Following this is a discussion of the contributions and limitations of the research. Then the thesis concludes with a summary of the findings and a discussion of future work.

This chapter has the following structure:

- Answering the research questions
- Discussion
- Contributions to theory
- Contributions to practice
- Limitations of the research
- Summary of findings
- Future work

6.1 ANSWERING THE RESEARCH QUESTIONS

The research questions guiding the study were presented in full at the end of Chapter 2 following on from the conceptual framework. Based on the study findings, the following sections answer the two principle research questions along with each of their contributing sub-questions.

The first research question is, *How is coordination achieved when using an agile development approach?* This section first answers each sub-question related to this research question, and then addresses the first research question in an overall manner.

RQ 1a What dependencies are present in an agile development approach?

A dependency is a potential or actual constraint on action in a situation. Knowledge, resource, and task dependencies are the major categories of dependency found in the projects in this study, agile and non-agile alike. These dependencies are called secondary dependencies since they were identified by grouping primary dependencies by their common properties. Dependency categories were defined during the case analyses in Chapter 4, and all definitions are reproduced in Appendix D Glossary of Terms.

Secondary dependencies found in the cases are summarised in Table 53. The percentage of coordination mechanisms used to address each dependency is shown, and an average value is calculated to indicate the relative predominance of knowledge dependencies found in the analyses. Knowledge dependencies are predominant in all

projects, averaging 72% of all dependencies per project. It is important to note that these numeric values are indicative only because they are based on a qualitative assessment.

Table 53 Comparison of secondary dependencies

	Cases				
Dependency	Land	Storm	Silver	Rock	Average
Knowledge	82	70	75	64	73
Task	9	19	16	24	17
Resource	9	11	9	12	10
Key All values are percentages indicating the relative number of coordination mechanisms used on the project to address the dependency, e.g. In project Land 82% of coordination mechanisms were addressing knowledge dependencies					

The frequently occurring dependencies in Storm, Silver, and Rock are resource dependencies. Both task and resource dependencies are relatively infrequent in Land. Since one of the selection criteria for cases in this study was that they were of some importance to the organisation where they occurred, it is likely they were resourced adequately, or resourced well, and this might explain the relatively low incidence of resource dependencies.

In Storm, Silver, and Rock, task dependencies are somewhat more numerous than resource dependencies. In the agile projects, it is possible that the higher number of people directly involved in development on the teams (i.e., Land 3, Silver 6, Storm 10 people respectively) increased the likelihood of task dependencies occurring (i.e., Land 9, Silver 16, Storm 19 coordination mechanisms used to address task dependencies respectively). This could be because more people performing different, but related tasks might lead to an increase in task dependencies. Evidence for this relationship is tenuous and needs further investigation to clarify any such relationship.

Task dependencies in Rock (24% of coordination mechanisms address a task dependency) are more numerous than in the agile projects due to strict role divisions. These divisions meant certain team members were constrained to wait for task completion by team members with different roles before proceeding with their assigned tasks. For example, Host developers would wait for middleware developers, and testers would wait for both groups to finish their tasks before beginning integration and testing. This is typical of a waterfall process, which was how that project team described their process.

Primary dependencies are summarised in Table 54, which shows the dependencies found during analyses of case data in Chapter 4, and the percentage of coordination mechanisms addressing those dependencies. Primary dependencies occurring in all projects include requirement, expertise, task allocation, activity, and entity dependencies. Dependencies occurring in only some projects include historical dependency, business process dependency, and technical dependencies.

Table 54 Comparison of primary dependencies

		Land	Storm	Silver	Rock
Knowledge	Requirement	50	40	18	26
	Expertise	9	15	23	21
	Task allocation	23	11	14	12
	Historical	-	4	20	5
Task	Activity	4.5	19	16	19
	Business process	4.5	-	-	5
Resource	Entity	9	2	2	2
	Technical	-	9	7	10
Key All values are percentages indicating the relative number of coordination mechanisms used on the project to address the dependency, e.g. In project Land 50% of coordination mechanisms were addressing requirement dependencies					

All projects had higher percentages of requirement and expertise dependencies than other dependencies. Since requirements are a critical input to software projects, it is unsurprising that they cause dependencies in projects. That is, when requirements are not readily available, this has a negative impact on project progress. This is because, whether the project is agile or non-agile, requirements emerge as the project progresses (this is expected practice in an agile project). In an agile project, when requirements fail to emerge in a timely manner this can cause the project to falter, and the project team must take some action to cope with the situation. This occurred in Silver and the project team chose to swap stories to maintain workflow and keep to the iteration schedule.

The percentage of coordination mechanisms for addressing expertise dependencies was high in Storm, Silver, and Rock. In Storm and Rock, this could be attributed to the type of system under construction. Each of these projects involved replacement of an existing complex system that had been in use in the organisation for many years. The project team would consult experts in the details of these systems so they could better understand their internal structure. This was to ensure they understood enough about the existing system to design the new system so it would integrate appropriately with existing infrastructure. Silver was somewhat different. In that project, expertise was

unevenly distributed among the project team members when the project began. Expertise therefore needed to be shared, and this situation led to a high number of expertise dependencies within the project.

Storm, Rock, and Silver also had historical dependencies because knowledge about decisions made when building the existing system needed to be understood in order to reproduce appropriate functionality in the new system.

The dependency analysis shows that acquiring information and knowledge is highly important in the projects, and lack of information affects project progress more than any other form of dependency. Consider the three dependencies that focus on information acquisition: requirements dependency, expertise dependency, and historical dependency. Summing the percentages of coordination mechanisms on all projects provides the following: Land 59%, Storm 59%, Silver 61%, and Rock 52%. These values show that at least half of the coordination mechanisms in use are for addressing dependencies related to obtaining information about requirements, or information on the structure and function of existing systems, or acquiring other people's expertise – either from other project team members or from people outside the project team.

To answer research question 1a, the dependencies present in the agile software development approach are knowledge dependencies, task dependencies, and resource dependencies. These dependencies can be decomposed as follows:

- Knowledge dependencies consist of requirements dependencies, expertise dependencies, historical dependencies, or task allocation dependencies. At least half the coordination mechanisms identified in the projects are used to address knowledge dependencies.
- Task dependencies can be activity dependencies or business process dependencies.
- Resource dependencies can be entity dependencies or technical dependencies.

There is no evidence in this study to suggest that dependencies in agile software development projects are different to those in non-agile projects.

Moving to address sub questions 1b and 1c of research question 1, consider Table 55. Information in this table is used in the following discussion. This table shows all coordination mechanisms identified in the four projects and those practices normally attributed to Scrum and XP, the two agile methods in use in the three agile projects.

Table 55 Coordination mechanisms and agile practices on all projects

Coordination strategy component		Coordination mechanism	Land	Storm	Silver	Rock	Scrum practice	XP practice
Synchronisation activities	P	Iteration zero for planning	✓				x	x
	I	Weekly iteration	✓				x	x
	I	Iteration planning session	✓				x	x
	D	Progress tracking with user stories	✓					x
	I	Story point prioritising	✓					x
	D	Daily team session (Ld) / daily stand-up (Sm, Si, Rk)	✓	✓	✓	✓	x	x
	I	Software release	✓		✓	✓	x	x
	P	Domain specialist on team (St) / technical specialist on team (Rk)		✓		✓		
	A	Assignment of story		✓			x	x
	I	Breakdown session (iteration)		✓			x	x
	A	Pair program (St) Help session (Si)		✓	✓			x
	A	Breakdown session (ad hoc)		✓				x
	A	Informal negotiation f2f (Si, Rk) Cross-team talk (Si, St)		✓	✓	✓	x	x
	A	Continuous build (Sm) / Continuous integration and test (Si, Rk)		✓	✓	✓	x	x
	P	Initial infrastructure sprint			✓		x	x
	I	Sprint planning/breakdown session			✓		x	x
	I	Two-week sprint			✓		x	x
	I	Retrospective			✓		x	
	A	Broadcast email			✓			
	I	Preparation for product demo			✓		x	x
	I	Product backlog maintenance			✓		x	
	A	Manual acceptance test			✓			x
	P	Initial meeting				✓		
	A	Informal chat using SMS				✓		
	A	Impromptu planning session				✓		
	A	Use case breakdown session				✓		
	A	Conference call with stakeholders				✓		
Key Ld – project Land Sm – project Storm Si – project Silver Rk – project Rock P - per project frequency I - per iteration frequency D - daily A - ad hoc ✓ - a coordination mechanism identified on the specified project x - an agile practice that appears as a coordination mechanism. Practices belonging to XP and Scrum are based on assessments by Strode (2005, Appendix K), and Williams (2010) BS – boundary spanning								

Table 51 continued: Coordination mechanisms and agile practices on all projects

Coordination strategy component		Coordination mechanism	Land	Storm	Silver	Rock	Scrum practice	XP practice
Synchronisation artefacts		Product backlog	✓	✓			x	
		Design specification document (living)	✓					
		Working software	✓	✓	✓			x
		User story	✓	✓	✓			x
		Burn-down chart	✓	✓				x
		Wallboard (public product backlog in Scrum)		✓	✓		x	
		Task		✓	✓		x	x
		Done list		✓	✓		x	x
		Whiteboard		✓				
		Wiki		✓				
		Code standards			✓			x
		Avatar on task (Si) Avatar on work item (Rk)			✓	✓	x	
		Rally (backlog & stories)			✓			
		Layered architecture			✓			x
		Source code control			✓	✓		
		Unit test suite			✓			x
		Ground rule			✓			
		Kanban wall				✓		
		Work item				✓		
		Work in progress limit				✓		
		JIRA™ (bug tracking)				✓		
		Architecture				✓		
BS activity	I	Workshop to generate backlog		✓			x	
	P	Nominated specialist end user		✓				
	I	User story prioritisation		✓	✓		x	x
	I	Software demo to user (St) Software demo to client (Si)		✓	✓		x	
	D	Dedicated time for consult		✓				
	A	Formal meeting		✓				
	A	Informal negotiation f2f		✓	✓	✓	x	x
	P	Acquisition of specialist knowledge		✓				
BS artefact		Request for change	✓					
		List of tests		✓				
		Test data spreadsheet				✓		
Availability		Single priority team (Ld, Si)/Fulltime team(Rk)		✓	✓	✓	x	x
Proximity		Team member co-location		✓	✓	✓	x	x
		Customer co-location		✓			x	x
Substitutability		Redundant skill (Ld, Si, Sm)/Specialisation(Rk)	✓	✓	✓	✓	x	x
Coord. role				✓		✓	x	
Impersonal artefact		Use case specification				✓		
		System design specification				✓		
		Unit test document				✓		
		Defect document				✓		

RQ 1b What coordination mechanisms are in use in an agile development approach?

Table 55 shows multiple coordination mechanisms are in use, many of them not recognised as agile software development practices.

Activities synchronising the team that are not recognised as agile practices include grafting a domain specialist onto the team, and using broadcast emails for serious system changes. Artefacts for synchronising the team include the use of a 'living' design specification document, whiteboards to hold design information, wikis for storing project and design information, the use of tools such as Rally™ to store product backlog information, using source code control, and setting commonly agreed ground rules for the team to follow. Boundary spanning activities, necessary when the customer or end-user group are not co-located or in close proximity to the team, include nominating a special end-user group to consult, setting up dedicated times for consultation, having formal meetings with end-users, and sending out team members for substantial time periods to acquire needed information from domain experts in other business units. Boundary spanning artefacts for communicating with customer or end-user groups include formal documents such as request for change forms, lists of acceptance tests, and specifications of test data in use.

This indicates that multiple coordination mechanisms are used in achieving agile software development. They include agile practices and other common sense practices. These practices are used in various combinations in different projects. It is only when looking at their contribution to coordination in a more abstract way that a discernable pattern emerges. The pattern identified in this study was named 'coordination strategy' and all agile projects followed this pattern, that is, project team members selected coordination mechanisms to achieve synchronisation, structure, and boundary spanning.

There are distinct differences in coordination mechanisms between agile projects and the single non-agile project, Rock. Impersonal artefacts were used in the non-agile project and not at all in the agile projects, and role specialisation was used in preference to substitutability. Role specialisation was a reflection of the system architecture imposed by the existing centralised IT system used in the organisation. This was a key focus around which coordination was organised. Project roles were strictly separated along architectural divisions (Host, middleware, front-end), the multiple coordination roles (four in this project) taken by project team members were

based on those roles. The architecture was also reflected in the layout of the Kanban wall (described in Appendix L).

In summary, the answer to the question of what coordination mechanisms are used in the agile approach is that multiple coordination mechanisms are used and they form three distinct categories:

- Synchronising coordination mechanisms that bring the project team together. These mechanisms include activities and artefacts. Activities occur per project, per iteration, daily, and ad hoc and these activities either produced or consume artefacts that also have a role in synchronisation
- Structuring coordination mechanisms that ensure project team members are available, located in close proximity, and can readily substitute for one another by performing each other's tasks
- Boundary spanning activities and associated boundary spanning artefacts, so that needed information and things are provided for the project.

RQ 1c What agile development practices act as coordination mechanisms in agile software development projects?

Table 55 shows that many agile practices act as coordination mechanisms in the three agile projects. The daily stand-up meeting was the single coordination mechanism that is a recognised agile practice, and this practice was used in all projects. Typically, daily meetings are not used in waterfall projects like Rock, but this team shared a single workspace with multiple other project teams who were using Scrum under the guidance of a coach. Rock chose not to use Scrum, but did adopt a Kanban wallboard, as opposed to a Scrum wallboard, along with daily stand-up meetings.

Substitutability (redundant skill), working software, and user stories are all agile practices acting as coordination mechanisms. Each of the three agile projects used these practices.

In summary, the answer to the question of what practices from the agile approach act as coordination mechanisms is that multiple agile practices have a coordinating function. Those used on all of the projects in this study were daily stand-up meetings, role substitutability, working software, and user stories. Iterations were also used, but at different durations. This small number of common practices is notable, and confirms research that finds large variations in the actual practices adopted in agile projects. This was discussed in the literature review in Chapter 2 in the section Agile Methods.

RQ1d How are coordination mechanisms related to dependencies in agile development projects?

One underlying assumption of this thesis is that coordination mechanisms address dependencies in a situation. Therefore, the relationship between coordination mechanisms and dependencies was analysed to identify any patterns.

In each case analysed in Chapter 4, a table was presented showing each coordination mechanism identified on the project along with its associated dependency (e.g. Tables 18, 27, 35, and 44). Those tables highlight four possible combinations of dependencies and coordination mechanisms. They are:

1. A single coordination mechanism addresses a single dependency.

A count of coordination mechanisms and those that address a single dependency provides the following figures (note: CM = coordination mechanism):

Land 13 single CM / 16 Total CM = ~80%

Storm 22 single CM / 33 Total CM = ~67%

Silver 22 single CM / 32 Total CM = ~69%

Rock 24 single CM / 30 Total CM = ~80%

These calculations indicate that most coordination mechanisms address a single dependency.

2. A coordination mechanism addresses multiple dependencies. This situation could be considered a form of efficiency in a project because a single practice manages multiple project dependencies. Certain coordination mechanisms addressed three or more dependencies as follows:

- Land: weekly iteration, iteration planning session, and product backlog.
- Storm: iteration story breakdown session, team member co-location, cross-team talk and informal face-to-face negotiation, the Scrum wallboard, and user stories.
- Silver: the Scrum wallboard

Note that Project Land did not use a wallboard because the physical layout of the work site did not allow for this type of shared wall space and half of the project team was not co-located. The team substituted this with a product backlog kept in a shared document. Therefore, this project achieved efficiency in coordination with weekly iterations involving a planning session with all the

team, recorded in a product backlog that could be referred to during the iteration by all team members.

3. A dependency is addressed by multiple coordination mechanisms. This situation implies either:
 - a. The dependency can only be addressed with multiple coordination mechanisms or,
 - b. Some coordination mechanisms may be redundant, or
 - c. Only one coordination mechanism is needed and can be substituted with an alternative coordination mechanism. This could be considered a form of flexibility in a project because any one of a number of different practices could be used to manage a project constraint.

Requirements dependency and expertise dependency are addressed by multiple coordination mechanisms. The findings in Tables 18, 27, 35, and 44 showed the following:

- Land: 11 coordination mechanisms of the 16 used in the project, address the requirements dependency
- Storm: 21 coordination mechanisms of the 33 used in the project, address the requirements dependency
- Silver: 10 coordination mechanisms of the 32 used in the project, address the expertise dependency, 9 addressed the historical dependency and 8 addressed the requirements dependency

In this study, it is only possible to report what coordination mechanisms address particular dependencies. Also, the reason why so many different coordination mechanisms were used to address the requirements dependency (i.e. in Land and Storm, but not in Silver) is not clear from the case data. Neither is it clear whether they were necessary, or whether they were adopted because that is the usual practice in Scrum or XP, and some of them may be unnecessary. Furthermore, it is also possible that these practices achieve other important purposes in addition to that of coordination. This provides an interesting problem for future investigation.

4. There are dependencies in a project that have no coordination mechanism counterpart, these dependencies are not managed and remain as unresolved constraints on a project. These constraints are those that can only be managed

by extending a project deadline or changing the nature of the final product by reducing its functionality or quality.

The dependencies that were unmanaged included:

- Land: requirements, activity
- Storm: requirements, activity, expertise, entity, technical
- Silver: requirements, activity

It is interesting that although the requirement dependency was addressed with multiple coordination mechanisms it was also the dependency that was, on occasion, unmanaged during the project lifecycle. This situation occurred when project progress was held up because detail about a requirement was not readily available from the client, customer, end-user, or their proxy. In Land, even with the customer proxy as part of the team, because the three people charged with that role were not co-located or available to the project team consistently, this caused problems with eliciting requirement details, just as occurred in Storm and Silver, with their external customers.

As shown in the list above, Storm had more types of dependencies than Land or Silver. This can be explained by the greater complexity of Storm, which had many more external parties involved in the project, was more technically complex than either of the other two projects, and also had a much greater number of integration points with other systems.

In summary, the first principal research question is, how is coordination achieved when using an agile development? This study has shown the following:

- Multiple agile practices act as coordination mechanisms in agile software development projects
- Agile software development practices do not provide all of the coordination mechanisms that a project may need, additional activities and artefacts may be required especially when the customer is not co-located or consistently available to consult with the project team.
- Coordination mechanisms in agile software development projects take three forms, all of which are needed on a project. They consist of coordination mechanisms for synchronisation, structuring, and boundary spanning.
 - Synchronisation coordination mechanisms have a frequency and agile projects use coordination mechanisms per project, per iteration, daily, and ad hoc.

- Structuring coordination mechanisms include those for availability, proximity, and substitutability.
- Boundary spanning coordination mechanisms and associated artefacts are needed in all projects but are more prevalent when the customer, end-user, client or their proxy is not part of the project team. Boundary spanning coordination mechanisms have a frequency and agile projects use coordination mechanisms per project, per iteration, and ad hoc. Boundary spanning may also include one or more people who take a coordinator role.

This has answered the first research question.

* * * * *

The second principal research question asks, *What is the relationship between the coordination strategy of an agile development approach and project coordination effectiveness?* This section first answers each sub-question related to this research question and then integrates these answers to answer this second research question.

In answering this research question, most of the results come directly from Chapter 5 where the theory of coordination was presented. Therefore, this section summarises much of that previously presented argument.

RQ 2a What is project coordination effectiveness in the context of agile software development?

Coordination effectiveness is one of the two main concepts of the theory of coordination. In Chapter 5 this definition was developed:

Coordination effectiveness occurs when the entire agile software development team has a 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 individuals 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.

This definition comes from analyses of case evidence, but is foreshadowed in research on coordination in agile software development and coordination more generally. For example, in a case study exploring what makes agile software development ‘work’, Pries-Heje et al. (2011) report that Scrum helped a project team “*to understand very clearly what work needed to be done within the whole project and the specific Sprint; what they were expected to do themselves, what others were doing, and how to*

coordinate work" (Pries-Heje & Pries-Heje, 2011, p. 24). Furthermore, Crowston and Osborn (2003) identify 'right place', 'right time', and 'right thing' as elements in their typology of dependencies and coordination mechanisms. The difference between that study and the findings from this study is Crowston and Osborn propose that these three elements are forms of dependency, whereas this study, based on grounded data, found them to be outcomes of a coordination strategy.

RQ 2b How do different coordination strategies contribute to agile software project coordination effectiveness?

Each project explored in this study used a different coordination strategy. Each project was found to use a different combination of primary coordination mechanisms from all of the other projects. As discussed in the development of the theoretical propositions in Chapter 5, the following relationships were observed. The argument for proposition 2 showed that synchronisation influences implicit coordination effectiveness. The argument for proposition 3 showed that structural coordination mechanisms influence implicit coordination effectiveness, and proposition 4 argued that boundary spanning coordination mechanisms influence explicit coordination effectiveness.

This research question was not fully answered in the study. The relationship might be better explored using quantitative methods of analysis to investigate variations in coordination strategy, and the impact of these variations on a project coordination effectiveness construct.

RQ 2c How do project complexity and project uncertainty influence the relationship between coordination strategy and coordination effectiveness?

This research has shown that complexity and uncertainty act as antecedents to coordination strategy because they influence the choice of coordination mechanisms employed, or the way they are used, and thus they influence a projects coordination strategy. Since coordination strategy influences coordination effectiveness the relationship between complexity and uncertainty, and effectiveness is indirect.

Project complexity influenced the frequency of iteration in Storm. Increasing the frequency of iteration increases coordination by synchronisation as more synchronisation activities are carried out in a certain time frame (e.g. carry out a task breakdown session each week instead of fortnightly). When project complexity was a problem in Storm, synchronisation activities were increased to better cope with this complexity by reducing the length of breakdown meetings and the number of chunks of functionality addressed during each iteration. This in turn meant the team were

working on less functionality simultaneously, thus simplifying daily work and maintaining coordination effectiveness.

When project uncertainty is high, and changing iteration frequency is not desirable or possible, this can be handled by changing the priority of stories. This maintains project coordination effectiveness because work can progress on a substitute story. The overall project outcome however, may be a less satisfactory product since lower priority stories result in a product with functionality of less importance to the customer.

When project uncertainty is high, and the customer is external to the project team, this can be handled by increasing the number and or frequency of boundary spanning coordination mechanisms. This will maintain project coordination effectiveness.

The second principal research question asked, what is the relationship between the coordination strategy of an agile development approach and project coordination effectiveness. First, the overall relationship between coordination strategy and coordination effectiveness is stated as follows:

An agile coordination strategy comprising synchronisation, structure, and boundary spanning components, leads to coordination effectiveness engaging both implicit and explicit components.

The detail of the relationship between coordination strategy and coordination effectiveness was expressed in a series of propositions that were elaborated in Chapter 5. These propositions also included the effect of various antecedents of coordination strategy. The propositions are repeated here:

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

Proposition 1b. *When the customer is an external party to the project, a coordination strategy that includes synchronisation, structure, and boundary spanning coordination mechanisms improves project coordination effectiveness. 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 2. *Synchronisation activities at all frequencies – project, iteration, daily, and ad hoc, along with their associated synchronisation artefacts, increase implicit coordination effectiveness.*

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

Proposition 4. *Boundary spanning coordination mechanisms including boundary spanning activities at all frequencies – project, iteration, and ad hoc, their associated boundary spanning artefacts, and a coordinator role increase explicit coordination effectiveness.*

Proposition 5. *To reduce project complexity increase the frequency of iteration and ad hoc synchronisation activities. The production of related synchronisation artefacts must be adjusted accordingly.*

Proposition 6. *Under conditions of project uncertainty, to maintain synchronisation activity frequency and production of associated artefacts, change the priority of stories.*

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.*

Proposition 9. *To reduce project complexity when the customer is not part of the team, introduce a coordinator role.*

This has answered the second principal research question. The following sections discuss the contributions of the study to IS theory, and to IS practice.

6.2 DISCUSSION

The findings in this study are reflected in some of the literature on agile software development, and on coordination. Three studies of agile software development support the findings in this study. Firstly, Hoda et al. (2010) identified a coordinator role in their formal grounded theory study of agile project management. The coordinator “acts as a representative of the self-organizing agile team to coordinate communication and change requests from customers” (ibid, p.5). The theory of coordination presented in this thesis also has a coordinator role. For example Storm had a tester who facilitated coordination between the team and the end-users, and Storm and Land had a project manager who facilitated coordination between the team and other business units in the organisation. In the theory developed in this thesis, the role of coordinator is more broadly defined than it is in Hoda et al.’s (2010) theory. In this thesis the coordinator role is a coordination mechanism that occurs when someone

on the project team initiates and sustains reciprocal interaction with other external parties to the project to meet project goals. This means that coordinator roles involve more than change requests, but includes any form of boundary spanning activity (e.g. arranging testing, organising visiting experts to join the team). Although the idea of a coordinator is conceptualised somewhat differently in Hoda' et al.'s study to this study, the two sets of findings support one another. A coordinator role clearly emerged within the agile projects in both sets of studies. This role is not unique to agile software development projects, however, because it also occurred in project Rock.

Another study supporting the coordination theory developed in this thesis is that of Pries-Heje and Pries Heje (2011). They identified coordination as a critical element explaining the effectiveness of Scrum in the context of a single distributed software development project. Their study found backlogs, wallboards, and daily meetings were practices for achieving coordination. In the coordination theory developed in this thesis, a daily meeting is conceptualised as a synchronisation activity (an activity involving the project team meeting together at the same time). Backlogs and wallboards were considered synchronisation artefacts produced and used during these synchronisation activities. Therefore, the findings of this thesis concur with those of Pries-Heje and Pries Heje (2011).

A third study of agile software development supporting the findings of this thesis is that of Moe, Dingosyr, and Dyba (2010). They found that "*not knowing what others were doing*" (Moe, et al., 2010, p. 488) interfered with successful team coordination in the context of a single co-located Scrum project. This finding is precisely what was found in this thesis, where one component of the coordination effectiveness concept is named "*know who is doing what*" (see Figure 22).

In Chapter 2 Literature Review, perspectives on coordination were identified as structural, architectural, technological, spatial, cognitive, coordination by artefacts, coordination by activities, temporal, and relational coordination. Some, but not all, of these literatures are reflected in the findings of this study.

Structural coordination reflects the structure of an organisation (Hatch & Cunliffe, 2006; Mintzberg, 1980). One form of structural coordination is mutual adjustment, which involves high levels of group coordination mechanisms (primarily meetings). Group meetings impose a high coordination cost (Van de Ven et al., 1976). Synchronisation activities in the coordination theory developed in this thesis are primarily achieved using group meetings (e.g. daily stand-ups, planning meetings, ad hoc meetings, and retrospective meetings) supporting the idea that agile software

development has high coordination cost. This thesis does not explore the coordination cost of agile software development, which might be a question for future research, i.e. does agile software development impose higher coordination cost than other forms of software development?

Architectural coordination is coordination imposed by the system architecture on the interactions of the development team (Herbsleb & Grinter, 1999). Architecture is considered a coordination mechanism in software engineering (Herbsleb, 2007) and in agile short cycle time development (Baskerville and Pries-Heje, 2004). This thesis found no link between architecture and coordination, although software developers who would be most aware of constraints imposed by the architecture, were intensively interviewed in each project. This does not mean architecture had no coordinative effect, merely that no evidence could be found in the data to support this idea.

Technological coordination, which is coordination achieved using software tools, also had no impact on coordination. Little research links tool use with agile project coordination, although an ethnographic study by McKenzie and Monk (2004) found concurrent versioning systems enables coordination. Although many different tools were in use in the projects explored in this thesis none of them were found to influence coordination. Similar to the case of architecture, this does not mean tools had no effect merely that no evidence could be found to support this idea.

Spatial coordination is achieved through the arrangement of artefacts or actors. In the co-located projects in this study, proximity was a coordination concept identified in all projects. Close proximity was achieved through spatial arrangements because in three projects the team members were all seated together in a project room and could overhear and see each other's activities. There is little research exploring the impact of proximity on agile software development, although the lack of proximity is often cited as problematic when teams are distributed within a building, or more widely (Agerfalk, Fitzgerald & Slaughter, 2009; Reifer, Maurer, Erdogmas, 2003). This thesis therefore provides evidence that spatial coordination, called proximity here, contributes to coordination in agile software development projects.

Cognitive coordination occurs within groups without explicit speech or message passing. The coordination effectiveness concept developed in this thesis has an implicit component (i.e. know why, know what is going on, know what to do, and know who is doing what). This component seems related to the cognitive coordination discussed in teamwork literature (see 2.2.4 Coordination in Teamwork Studies) which includes shared mental models, expertise coordination, and mindfulness. Although, Matook and

Kautz (2008) studied agile projects from a mindfulness perspective, there are few studies of this aspect of agile software development. This thesis provides support for the importance of cognitive coordination in agile software development projects.

Coordination by artefacts is coordination mediated by artefacts in a work process. The coordination theory developed in this thesis has two forms of coordinative artefact: synchronisation artefacts and boundary spanning artefacts. Support for the coordinative properties of synchronisation artefacts, such as the wallboard, is provided by Ren, Kiesler, and Fussell (2008) who found that publicly visible whiteboards are used for coordinating activities within hospitals. Sharp and Robinson's (2008) ethnography of an XP project also supports the idea that wallboards and story cards are coordinative artefacts. Boundary spanning artefacts developed in this thesis are similar to the boundary objects discussed in the IS literature by Levina and Vaast, 2005. They define a boundary object as any physical artefact such as a prototype, design drawing, or use case used to achieve boundary spanning (interaction between distinct groups). Although these authors do not explicitly link boundary objects with coordination, they imply this because their boundary objects are used to communicate between distinct business groups. This provides support for the idea proposed in this thesis that boundary spanning artefacts support coordination.

Coordination by activities is coordination achieved by group action. The ethnographic studies of agile software development identified frequent meetings, pair programming, and refactoring, and daily stand-up meetings as coordinating activities (Moe, et al., 2008; Robinson & Sharp, 2004; Sharp & Robinson, 2004; Sharp & Robinson, 2008). The coordination strategy in this thesis refines this idea: coordinative activities are for synchronising a team (bringing them together at the same place and time), or for boundary spanning (linking to another external group).

Bardram (2000) defined temporal coordination as an activity that ensures collaborative work can be achieved at the appropriate time. In this thesis, the concepts of synchronisation activity and boundary spanning activity embody temporal coordination. Synchronisation and boundary spanning activities occur regularly following the particular iteration cycles used in a project.

Relational coordination is coordination mediated by people's assigned roles, rather than between individuals or tasks (Gittell, Seidner, & Wimbush, 2010). The substitutability component of the coordination strategy concept in this thesis could be considered a form of relational coordination. Substitutability involves reducing or eliminating role differentiation to support effective coordination. The agile project

teams in this study made special efforts to remain flexible by reducing role differentiation. Rock, the non-agile project, was the project where roles were found to influence coordination, leading to the identification of the coordinator role.

Multiple different perspectives on coordination exist in the literature. As this discussion has shown, many are reflected in the coordination theory developed in this thesis.

6.3 SUMMARY OF FINDINGS

This research provides insight into coordination in agile software development projects. The literature review identified a variety of different perspectives on coordination. The empirical research presented in this thesis found specific dependencies present in software development projects along with coordination mechanisms for addressing them.

Coordination concepts

The thesis began by reviewing literature on two topics, the agile software development approach from its inception to the current date, and coordination in the fields of organisation science, IS projects, and teamwork. Coordination Theory was summarised to show how it could contribute to a study of software development practice. The literature review found a variety of different perspectives on coordination. Coordination can be temporal, relational, cognitive, structural, architectural, technological, spatial, and achieved by coordination artefacts and by coordination activities. The literature review also summarised quantitative studies linking coordination and software projects, and found coordination contributes to project success and team success.

Dependencies

A dependency analysis of three co-located agile software development projects and one non-agile project provided a second group of findings. Coordination Theory provided the fundamental idea that dependencies lead to potential or actual constraints on action in a situation, and coordination mechanisms exist to address these dependencies. This study found both agile and non-agile projects address the same secondary dependencies: knowledge dependencies, task dependencies, and resource dependencies. Knowledge dependencies include requirements dependencies, expertise dependencies, historical dependencies, and task allocation dependencies. Knowledge dependencies were found to account for more than half of the dependencies within the projects. The remainder of dependencies are task dependencies, which include activity

and business process dependencies, and resource dependencies, which include entity and technical dependencies.

Coordination strategy

A coordination strategy is apparent in agile software development projects and this strategy differs from that of non-agile projects. An analysis of coordination mechanisms in the agile software projects found multiple agile practices act to achieve coordination in conjunction with other practices, not specified in agile methods. These mechanisms form a generic strategy which was found to include three main categories of coordination mechanism: synchronisation, structure, and boundary spanning.

Synchronisation mechanisms bring the whole team together physically and simultaneously at regular intervals. At these synchronisation points (e.g. daily meetings, sprint planning meetings, retrospectives) the team works together on project activities and shares decision-making and knowledge about the project. The frequency of these intervals is once per project, per iteration, daily and ad hoc (i.e. as and when required). During these activities, artefacts (e.g. stories, tasks, wallboards, done lists, ground rules, progress graphs) are generated and used to not only store project information (e.g. requirements, design decisions, project progress) but also to guide project participants in their work (e.g. tasks to carry out). These artefacts tend to be temporary, although some are stored permanently in electronic form.

Structuring mechanisms are a second category of coordination mechanism. There are three types - close proximity, constant availability of all team members to one another, and substitutability, which means individuals have the ability to perform each other's work to maintain project progress.

Boundary spanning mechanisms are the third category of coordination mechanism. These mechanisms are used to interact with the wider organisation, and other organisations, to ensure resources and information are acquired as and when they are needed. Boundary spanning is achieved with activities at the frequency of once per project, per iteration, and ad hoc. Boundary spanning artefacts may be produced, and a coordinator role may be necessary. Boundary spanning is achieved by single project team members, and typically does not involve the whole project team acting together. Boundary spanning coordination mechanisms were found to increase when the customer is not a full participant in the team, that is they are not co-located with the project team, consistently available, and taking part in synchronisation coordination mechanisms.

Coordination strategy in the non-agile project differed in four ways from the three agile projects. The first difference between the agile projects and Rock was that Rock did not use iteration. This considerably changed the frequency of their synchronisation activities, which were once per project, daily, and ad hoc. Without iteration, the number of synchronising activities is reduced (e.g. with a one-week iteration, the whole team meets at least 12 times in a 3 month project. Without iteration, these meetings might not occur at all). A second difference was that Rock used impersonal artefacts for coordination (e.g. permanent documents created to define requirements, system design, and tests). A third difference was role specialisation. Role specialisation was the norm in Rock, whereas the agile projects chose substitutability. The final difference was that, in Rock, boundary spanning activities were ad hoc only, whereas the agile projects chose to use boundary spanning activities at three frequencies: per project, per iteration, and ad hoc.

Rock was an atypical non-agile project in some respects. The team members were all seated together, and they chose to have a daily stand-up meeting beside their Kanban wallboard. They were influenced to adopt these practices because they were located alongside eight Scrum projects that were under the guidance of a Scrum consultant and coach. This meant that the analysis of Rock showed the same proximity and availability profile as the agile projects. They also had a daily synchronisation activity. These coordination mechanisms might not be typical of other non-agile projects, particularly in the use of a daily meeting.

Efficient agile practices

Agile practices that address more than a single dependency can be considered efficient practices, according to the arguments presented when addressing research sub-question RQ1d (page 263). The following practices were found to address three or more dependencies: short iterations of one or two weeks, iteration planning sessions, a product backlog, team member co-location, cross-team talk and informal face-to-face negotiation, the Scrum wallboard, and user stories.

Project complexity and uncertainty, and organisational structure

Project complexity, project uncertainty, and organisation structure were found to act as antecedents to a project's coordination strategy. Project complexity led to two observed changes. First, a coordinator role or roles was introduced into the project to ensure an adequate supply of resources and knowledge to the project, which are necessary to maintain project progress. Second, iteration frequency was changed, to

shorten the duration of the iteration. This increased overall iteration-level coordination mechanisms but reduced the number of open stories being worked on concurrently. A smaller number of open stories in an iteration reduces both the time duration of story breakdown sessions, and simplifies the work of the team because they work on less tasks simultaneously, thus simplifying their daily work.

Project uncertainty, related to uncertainty about requirements, led one project to change the priority of stories to maintain project progress. Another effect of high project uncertainty, when the customer is external to the team, is to increase the number or frequency of boundary spanning coordination mechanisms to maintain project progress.

Organisation structure was found to influence proximity and availability. Matrix organisation structures reduce proximity and availability of project team members to one another because individuals work on more than one project simultaneously.

Coordination effectiveness

The study found clear evidence for a link between coordination strategy – the actions and artefacts used in a project – and coordination effectiveness. Coordination effectiveness was found to be composed of two components according to the perceptions of the research participants. Implicit coordination effectiveness is achieved when the project team understands 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 member's work. Explicit coordination effectiveness is achieved when 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.

Although there are distinct differences in the coordination strategy of the agile and non-agile projects, all project participants perceived coordination effectiveness in a similar way. This suggests that the coordination effectiveness concept defined in this study is applicable to both agile and non-agile contexts.

6.4 CONTRIBUTIONS TO IS THEORY

This research contributes to information systems theory in various ways. Firstly, this thesis develops and presents a theory explaining coordination in co-located agile development projects. This theory draws on case study evidence and extant literature on coordination and agile software development, and is unique in the literature on

coordination and on information systems development. This is a mid-range theory according to Gregor (2006), who defined such theory as moderately abstract, of limited scope, and easily able to generate testable hypotheses. The theory of coordination developed and presented in this thesis is moderately abstract in that coordination effectiveness is an abstract concept that cannot be measured directly. That is not true of the coordination strategy concept, which is a concrete concept grounded in actual practices used in software projects. The scope of this theory is limited to co-located, small sized (6 to 10 people), agile software development projects using methodologies XP, Scrum, or near variants. The theoretical propositions developed to explain the relationship between the strategy and effectiveness concepts could readily be converted to testable hypotheses.

The second contribution of this theory of coordination is the use of Malone and Crowston's (1994) Coordination Theory to identify coordination mechanisms in a situation, and use this as a basis to build a theory of coordination unique to agile software development. This theory of coordination relevant to agile software development includes a coordination outcome concept named coordination effectiveness. Coordination theory does not have such a concept; therefore the coordination theory in this study can be considered an extension to Coordination Theory. The basic assumption of Coordination Theory is that coordination mechanisms address dependencies in a situation, but Coordination Theory makes no claims about what effective coordination might be other than to propose it implies that dependencies in a situation are managed. The theory developed in this thesis provides a detailed concept of coordination effectiveness which is conceptualised as two components: explicit coordination and implicit coordination. Coordination effectiveness in this theory is an outcome of some cohesive coordination strategy made up of coordination mechanisms for synchronisation, structure, and boundary-spanning.

The third contribution to theory is that this theory of coordination in agile projects provides an instance of the more general theoretical framework proposed by Espinosa, Lerch, and Kraut (2004). Their framework was formed from argumentation and evidence from geographically distributed, asynchronous teams. This theory of coordination in the agile context is for co-located and synchronous teams. Therefore, the theory presented in this study extends Espinosa, et al.'s theory into the co-located agile software project context. In addition, this study defines the concepts of coordination strategy and coordination effectiveness more precisely than their framework does.

Another contribution of this theory is that it provides a base theory that might be extended in future work to explain coordination in the context of distributed agile software development.

A final contribution is to IS project success literature. Project success has many antecedents (as discussed in section 2.2.7 Project success and coordination, and shown in Table 5). One important antecedent is project coordination (Curtis et al. 1988; Nidumolu, 1995). This study has deepened and extended our understanding of the way coordination is achieved, and consequently contributes to our understanding of the achievement of project success.

Not only does the theory in this thesis contribute to existing IS theory by original contribution, and by extension of existing theories of coordination, but it makes potential contributions to some IS reference fields (Baskerville & Myers, 2002). Agile software development is an important area of research in software engineering, as discussed in Chapter 3 Literature Review, and the theory presented has relevance there. This is discussed more specifically in the next section on contributions to practice. This theory can also contribute to teamwork studies where coordination is considered a central phenomenon and a manifestation of team cognition (Fiore & Salas, 2004). The coordination effectiveness concept could contribute to the teamwork field. The concept of coordination effectiveness could also contribute to organisation theory because, in that field, implicit and explicit coordination in work teams is of interest because they are known to contribute to team performance (Rico, et al., 2008). Furthermore, the field of organisation science also has an emerging sub-field focusing on practice theory (Feldman & Orlikowski, 2011), and this study also contributes meaningfully to that research stream. Agile software development is of interest in the field of project management because of its impact on the project manager role, the management of software projects, and its proposed adoption in general project management (Levitt, 2011). Furthermore, project management is a research area that typically draws on organisation theory for its ideas on coordination (McBride, 2008). This study provides to the project management field not only an understanding of coordination in agile software development projects, but also a concept of coordination effectiveness more nuanced and appropriate than organisation theory coordination concepts provide.

6.5 CONTRIBUTIONS TO IS PRACTICE

This research makes contributions to IS practice. One contribution is the provision of four detailed case descriptions describing agile and non-agile software development projects. Each case description covers four areas: project detail, team detail, development methodology, and project issues. One participant on each project verified the accuracy of these descriptions to ensure they reflect actual project practice and issues. The value of these descriptions is in their content. Three of these descriptions describe how agile software development is carried out in typical organisations developing typical software products. Practitioners can use these cases to learn how others carry out agile software development and the project issues that can occur when using this development approach.

Another contribution for practitioners comes from the conceptualisation of coordination strategy. In its current form, this concept can be used directly by practitioners when they make decisions about which agile practices to adopt. 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 & Fitzgerald, 2007b). It is also common to find practitioners combining practices from two or more methods (Fitzgerald, et al., 2006). This was discussed fully in the literature review. As a practical matter, when practitioners select practices, either from a single agile method or from multiple agile methods, they can use the coordination strategy concept as an aid in identifying which agile practices to choose to achieve coordination coverage. The coordination strategy concept suggests practices should be selected that provide synchronisation at all of the identified frequencies: project, iteration, daily, and ad hoc. Artefacts can be produced and consumed at all these frequencies because they also contribute to coordination. In addition, practitioners should consider whether their organisational structure will ensure proximity and availability are achievable for all team members. Substitutability should be encouraged and supported, 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 solves 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 in the findings 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.

Efficiency is a concern in software projects, and a coordination mechanism that addresses multiple dependencies in a project might contribute to project efficiency, which occurs when a single practice manages multiple project dependencies.

Practitioners are advised to use the following practices to enhance efficiency: iterations of one or two weeks, iteration planning sessions with story breakdown, a product backlog, a Scrum wallboard, and stories. Further, practitioners should implement project team member co-location, and they should support cross-team talk and informal face-to-face negotiation. Each of these practices was found to address multiple project dependencies.

Practitioners can use the coordination effectiveness concept as a guide for qualitatively assessing the coordination effectiveness of a project at various time points. Although practitioners cannot use the concept of coordination effectiveness for precise measurement in its current form, once operationalised it has a number of uses. The next section on future work discusses these uses.

Another contribution of this research is in the identification of dependencies common to agile and non-agile projects. These dependencies include knowledge, task, and resource dependencies. Knowledge dependencies are requirement, expertise, task allocation, and historical dependencies. This study finds that knowledge dependencies make up more than half of the dependencies a project encounters. This means that constraints on knowledge, that is not having ready access to needed information, is a major constraint on project progress. Therefore, practitioners are advised to pay close attention to these dependencies in order to enhance their project coordination.

6.6 LIMITATIONS OF THE RESEARCH

The limitations of this study include those common to qualitative case study research and those peculiar to this research study. The limitation of qualitative case study research is its lack of generalisability (Yin, 2003b). Findings cannot be generalised to

the wider population of all agile software development projects since random sampling is not part of the methodology, nor are large numbers of cases studied. The aim of qualitative research is not to generalise in this way, but to provide understanding, explain the complex nature of a situation, and to generalise to theory. Rather than statistical sampling, theoretical sampling is used in case study research, which means selecting cases that highlight particular characteristics in a situation that can contribute to concept definition and explain relationships (Eisenhardt & Graebner, 2007). In this study, three typical cases of agile software development were selected according to specific criteria (described in Chapter 3 Research Methodology) to provide evidence for the theory developed in this study, and another project was selected to provide evidence that the theory holds less well in a non-agile context. Therefore, although this coordination theory is tentative, and requires testing using quantitative research methods to provide statistical generalisation, the study was carried out with rigor following accepted guidelines for case study research. These guidelines for rigor contribute to the veracity of the findings and resultant theory. Furthermore, the theory draws not only on case evidence, but also on empirical and theory-based coordination literature making the theory somewhat more robust than if it were based solely on case-based reasoning.

Limitations specific to this research study are in theory development, case selection, data collection, and data analysis phases. The theory developed in this study makes a presumption of causality, that is, a coordination strategy *leads to* a level of coordination effectiveness. Although some evidence supports this relationship, its existence is tentative and needs additional supporting evidence. Furthermore, the relationship between organisation structure and strategy, complexity and strategy, and uncertainty and strategy was not found in all cases but only some cases. Therefore, additional investigation of this aspect of the theoretical model is called for.

Another limitation of the theoretical model is the lack of evidence explaining interaction effects between synchronisation, structure, and boundary spanning coordination mechanisms. In its current form, the theory assumes they are all of equal importance in contributing to coordination effectiveness. Likewise, at the level of individual coordination mechanisms, the evidence for substitutability of different coordination mechanisms addressing the same dependency was not explored, therefore no evidence for this was found. Furthermore, interaction effects between implicit and explicit components of coordination effectiveness were not explored,

although teamwork coordination research makes a case for tradeoffs between them (Espinosa, et al., 2004).

There is insufficient evidence in the study to support an understanding of the effect of variation in the values of the coordination strategy (i.e. different combinations of coordination mechanisms) concept on coordination effectiveness. It is possible that different coordination strategies lead to similar levels of coordination effectiveness, indicating equifinality is possible in this theory.

The theory is empirically grounded in projects based solely in Wellington, New Zealand. Although this research was conducted in the New Zealand context, seven of the 15 people interviewed were born and professionally educated in other countries before immigrating to New Zealand. Therefore, the findings may be applicable internationally to projects with a similar profile to those in this study, since the experiences and training of the participants is likely to be reflected in their responses to the interview questions.

A further limitation is the applicability of the theory to all forms of agile project. The theory is based on evidence from agile projects using Scrum and XP practices, which may have less relevance for practitioners using one of the many other agile methods. The literature review showed that XP and Scrum are currently the dominant agile methods, so this might not be a major limitation.

The data collection process also suffered from limitations. The number of participants was small; the findings might have been strengthened with a larger number of cases. Furthermore, wider polling of case study participants, especially end-users, customers, or their proxies may have provided different perspectives on dependencies and coordination mechanisms.

Another limitation is in the design of the data collection procedure. Data were collected during interviews and at certain points in time. The participants were recalling past events, both recent (for example, what happened yesterday) and in longer projects, events that occurred many months before. Therefore, it is possible that all coordination mechanisms and all dependencies were not captured during data collection. To mitigate this, at least one participant in each case was asked to recount the overall project process and history in order to capture potential dependencies and coordination mechanisms for later exploration during interviews of project team members.

In the analysis phase limitations include a bias towards evidence from interview data because it provided richer and more readily available detail. Observations, while they are very informative, were not easy to organise and were time consuming to conduct, therefore observation made up only a small proportion of the data collected. Another limitation of the analysis was the lack of member checking of the analytic coding. This involves having the analytic coding of data checked by other independent people who are not involved in the research. This was not possible because people with the required expertise who had the time and motivation to do this member checking could not be located. Therefore only the sole researcher checked data coding.

Overall, there were a number of limitations in the study. Not all of them were mitigated during the research process. Further cases or the use of other research methods to verify the findings of this study should address these limitations.

6.7 FUTURE WORK

Further work would be useful to confirm the efficacy of the theory presented in this thesis and refine its concepts and proposed relationships. Future research should aim to verify the concept of coordination strategy (i.e. certain agile practices acting as coordination mechanisms), and in particular the relative contribution of synchronisation (especially at different iteration durations), structure, and boundary spanning coordination mechanisms, along with any interaction effects between these coordination components. Further verification of the concept of coordination effectiveness, in particular the contribution of, and interaction between, implicit and explicit coordination especially at different phases of a project and under various project conditions is needed. The relationship between coordination strategy and coordination effectiveness, in particular the effect of different coordination strategies on implicit and explicit coordination effectiveness is another area for investigation. This would mean investigating and comparing agile projects using different agile practices with a coordinative function, and the impact of these different strategies on coordination effectiveness.

The concepts of project uncertainty and project complexity need better conceptualisation since, although they are defined in the literature, those definitions are not mutually exclusive in their current form. Clearly, project complexity may increase project uncertainty, although the reverse relationship seems unlikely. Precise definitions of complexity and uncertainty would enable robust studies of their impact on coordination strategy. This would lead to information for practice because, given

certain levels of project complexity and project uncertainty one could select some coordination mechanisms (i.e. certain agile and non-agile coordinative practices) in preference to others.

In future, it would be possible to test this proposed theory using experiments, simulation, or survey methods. Experimentation in this area is seldom used because of multiple factors involved in the software development process; likewise, simulation of individual coordination mechanisms, pair programming for example, would be extremely difficult. Survey methods could provide a useful and appropriate way to test this theory in the field.

Another useful area of future work would be to operationalise the coordination effectiveness concept. This would provide a valuable measure of coordination effectiveness in agile, and possibly other, software projects. The concept is not currently in a form that can be directly operationalised, but conversion would seem to be straightforward since the factors comprising the concept are simple. 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. It could assist organisations to identify and address their coordination problems in a timely fashion, and improve the likelihood of successful agile project completion. The concept could also be used to measure the effect of different coordination strategies, in particular different coordination mechanisms and combinations of coordination mechanisms, on coordination effectiveness. In this way, the most effective mechanisms could be selected for coordinating projects. Further use for this concept might be to compare the coordination effectiveness of distributed agile and co-located agile projects, agile and non-agile projects, and between different system development methodologies that provide different coordination strategies. In particular, approaches to development such as lean development with Kanban processes.

Another benefit of a precisely defined, operationalised, and empirically tested coordination effectiveness concept would be to test its contribution to project success or effectiveness. Project success is not yet well defined. Research such as this would make an important and long-lasting contribution to IS project literature.

* * * * *

This research contributes a theory of coordination to the information systems field that explains coordination in co-located agile software development projects, based on an extensive literature review and evidence from empirical data from case study research.

This study has found that agile software development embodies effective coordination by providing simple practices that work together to assist teams to manage the knowledge dependencies, task dependencies, and resource dependencies common in software projects. These simple practices provide a structure that supports the synchronisation necessary for shared team and customer understandings. In addition, boundary-spanning coordination mechanisms need to be used alongside agile software development practices to ensure the flow of necessary information and resources from the organisation into the project.

Future testing of this theory in the field would contribute to our understanding of information systems development by giving organisations and individuals involved in the high-risk task of system development a better understanding of agile software development, and how it contributes to the success of software projects.

* * * * *

This thesis has contributed to the following research publications:

- A conference paper presenting the coordination effectiveness concept (Strode, Hope, Huff, & Link, 2011).
- A journal article presenting the coordination theory with evidence from three cases (Strode, Huff, Hope, & Link, 2012).
- A conference paper proposing a taxonomy of dependencies in agile software development (Strode & Huff, 2012).

7. REFERENCES

- Abbas, N., Gravell, A. M., & Wills, G. B. (2008). Historical roots of agile methods: Where did "agile thinking" come from? In P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan & X. Wang (Eds.), *Proceedings of the 9th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP 2008, June 10-14, 2008, Limerick Ireland*, (Vol. LNBIP 9, pp. 137-146). Berlin, Heidelberg: Springer.
- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis. *VTT Publications 478*. Retrieved 1 January, 2004, from <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>
- Abrahamsson, P., Warsta, J., Siponen, M. K., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. *Proceedings of the 25th International Conference on Software Engineering, ICSE'03* (pp. 244-254). Washington, DC, USA: IEEE Computer Society.
- Adler, P. S. (1995). Interdepartmental interdependence and coordination: The case of the design/manufacturing interface. *Organization Science*, 6(2), 147-167.
- Agarwal, N., & Rathod, U. (2006). Defining 'success' for software projects: An exploratory revelation. *International Journal of Project Management*, 24(4), 358-370.
- Agerfalk, P. J., & Fitzgerald, B. (2006). Flexible and distributed software processes: Old petunias in new bowls? *Communications of the ACM*, 49(10), 27-34.
- Agerfalk, P. J., Fitzgerald, B., & Slaughter, S. A. (2009). Flexible and distributed information systems development: State of the art and research challenges. *Information Systems Research*, 20(3), 317-328.
- Akgun, A. E., Keskin, H., Byrne, J., & Imamoglu, S. Z. (2007). Antecedents and consequences of team potency in software development projects. *Information and Management*, 44(7), 646-656.
- Aladwani, A. M. (2002). An integrated performance model of information systems projects. *Journal of Management Information Systems*, 19(1), 185-210.
- Albino, V., Pontrandolfo, P., & Scozzi, B. (2002). Analysis of information flows to enhance the coordination of production processes. *International Journal of Production Economics*, 75(1-2), 7-19.
- Allen, R. E. (Ed.). (1990). *The Concise Oxford Dictionary of Current English* (8 ed.). Oxford: Clarendon Press.
- Ambler, S. W. (2008). *The AgileUP*. Retrieved from <http://www.ambysoft.com/unifiedprocess/agileUP.html>
- Ambler, S. W. (2009). *Agile adoption rate survey results: February 2008* Retrieved 20 July, 2011, from <http://www.ambysoft.com/surveys/agileFebruary2008.html>

- Amrit, C., & van Hillegersberg, J. (2008). Detecting coordination problems in collaborative software development environments. *Information Systems Journal*, 25(1), 57-70.
- Andres, H. P., & Zmud, R. W. (2001). A contingency approach to software project coordination. *Journal of Management Information Systems*, 18(3), 41-70.
- Aoyama, M. (1996). Beyond software factories: Concurrent development process and an evolution of software process technology in Japan. *Information and Software Technology*, 38, 133-143.
- Aoyama, M. (1998). Agile software process and its experience. *Proceedings of the 1998 International Conference on Software Engineering* (pp. 3-12). Washington, DC, USA: IEEE Computer Society.
- Argyris, C., & Schon, D. (1974). *Theory in Practice: Increasing Professional Effectiveness*. Oxford, England: Jossey-Boss.
- Arrow, H., McGrath, J. E., & Berdahl, J. L. (2000). *Small Groups as Complex Systems: Formation, Coordination, Development, and Adaptation*. Thousand Oaks, CA: Sage.
- Atkinson, R. (1999). Project management: Cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management*, 17(6), 337-342.
- Avgerou, C., & Cornford, T. (1993). A review of the methodologies movement. *Journal of Information Technology*, 5, 277-286. doi: 10.1057/jit.1993.38
- Avison, D., & Fitzgerald, G. (2006a). *Information Systems Development: Methodologies, Techniques and Tools* (4 ed.). London: McGraw-Hill.
- Avison, D., & Fitzgerald, G. (2006b). Methodologies for developing information systems: A historical perspective. In D. Avison, S. Elliot, J. Krogstie & J. Pries-Heje (Eds.), *The Past and Future of Information Systems: 1976-2006 and Beyond* (pp. 27-38). Boston: Springer.
- Avison, D., & Pries-Heje, J. (Eds.). (2005). *Research in Information Systems: A handbook for Research Supervisors and Their Students*. Amsterdam: Elsevier Butterworth Heinemann.
- Aydin, M. N., Harmsen, F., van Slooten, K., & Stegwee, R. A. (2005). On the adaptation of an agile information systems development method. *Journal of Database Management*, 16(4), 24-40.
- Baccarini, D. (1996). The concept of project complexity - a review. *International Journal of Project Management*, 14(4), 201-204.
- Bacharach, S. B. (1989). Organizational theories: Some criteria for evaluation. *The Academy of Management Review*, 14(4), 496-515.
- Bacon, C. J., & Fitzgerald, B. (2001). A systemic framework for the field of information systems. *The DATA BASE for Advances in Information Systems*, 32(2), 46-67.

- Bardram, J. E. (2000). Temporal coordination: On time and coordination of collaborative activities in a surgical department *Computer Supported Cooperative Work*, 9(2), 157-187. doi: 10.1023/A:1008748724225
- Barki, H., Rivard, S., & Talbot, J. (1993). Toward an assessment of software development risk. *Journal of Management Information Systems*, 10(2), 203-225.
- Barki, H., Rivard, S., & Talbot, J. (2001). An integrative contingency model of software project risk management. *Journal of Management Information Systems*, 17(4), 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(1), 25-44.
- Barthelmess, P. (2003). Collaboration and coordination in process-centered software development environment: A review of the literature. *Information and Software Technology*, 45(13), 911-928.
- Baskerville, R., Levine, L., Pries-Heje, J., Ramash, B., & Slaughter, S. (2001). How internet companies negotiate quality. *IEEE Computer*, 34(5), 51-57.
- Baskerville, R., & Myers, M. D. (2002). Information systems as a reference discipline. *MIS Quarterly*, 26(1), 1-11.
- Baskerville, R., & Pries-Heje, J. (2004). Short cycle time systems development. *Information Systems Journal*, 14(3), 237-264.
- Baskerville, R., Travis, J., & Truex, D. P. (1992). Systems without method: The impact of new technologies on information systems development projects. In K. E. Kendall, K. Lyytinen & J. DeGross (Eds.), *Transactions on the Impact of Computer Supported Technologies in Information Systems Development* (pp. 241-260). Amsterdam: Elsevier Science Publications.
- Beck, K. (1999). Embracing change with Extreme Programming. *Computer*, 32(10), 70-77.
- Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley.
- Beck, K., & Andres, C. (2005). *Extreme Programming Explained: Embrace Change* (2 ed.). Boston: Addison-Wesley.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001a). Manifesto for agile software development. Retrieved 1 January 2012, from <http://www.agilemanifesto.org>
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001b). Principles behind the agile manifesto Retrieved 3 March, 2009, from <http://agilemanifesto.org/principles.html>
- Beedle, M., Devos, M., Sharon, Y., Schwaber, K., & Sutherland, J. (1999). Scrum: A pattern language for hyperproductive software development. In N. Harrison, B. Foote & H. Rohnert (Eds.), *Pattern Languages of Program Design* (pp. 637-651). New York: Addison-Wesley.

- Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly*, 11(3), 369-386.
- Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): an empirical view. *European Journal of Information Systems*, 8(3), 211-223.
- Beynon-Davies, P., MacKay, H., & Tudhope, D. (2000). 'It's lots of bits of paper and ticks and post-it notes and things...': A case study of a rapid application development project. *Information Systems Journal*, 10(3), 195-216.
- Boehm, B. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61-72.
- Boehm, B., & Turner, R. (2004). *Balancing Agility and Discipline*. Boston: Addison-Wesley.
- Bostrom, R. P., & Heinen, J. S. (1977). MIS problems and failures: A socio-technical perspective. Part 1: The causes. *MIS Quarterly*, 1(3), 17-32.
- Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38(4), 275-280.
- Broza, G. (2004). Adapting extreme programming to research, development and production environments. In C. Zannier, H. Erdogmas & L. Lindstrom (Eds.), *Proceedings of the 4th Conference on Extreme Programming and Agile Methods, XP/Agile Universe 2004* (Vol. 3134 Lecture Notes in Computer Science, pp. 139-146). Berlin: Springer-Verlag.
- Burton-Jones, A., McLean, E. R., & Monod, E. (2004). In pursuit of MIS theories: Process, variance, and systems. Working paper, Department of Computer Information Systems, Georgia State University.
- Butler, B. S., & Gray, P. H. (2006). Reliability, mindfulness and Information Systems. *MIS Quarterly*, 30(2), 211-224.
- Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2004). How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. *Proceedings of the 37th Hawaii International Conference on System Sciences* (pp. 5-8). Hawaii: IEEE Xplore digital library. Retrieved from <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1265237&isnumber=28293>. doi: 10.1109/HICSS.2004.1265237
- Cao, L., & Ramesh, B. (2007, March April). Agile software development: ad hoc practices or sound principles? *IT Pro*, 41-47.
- Cataldo, M., Wagstrom, P. A., Herbsleb, J. D., & Carley, K. M. (2006). Identification of coordination requirements: implications for the design of collaboration and awareness tools. *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work CSCW '06*. New York, NY, USA: ACM.
- Cavaye, A. L. M. (1996). Case study research: a multi-faceted research approach for IS. *Information Systems Journal*, 6(3), 227-252.

- Charette, R. N. (2002). *Foundations of Lean Development: The Lean Development manager's guide* (Vol. 2). Spotsylvania, Va.: ITABHI Corporation.
- Checkland, P. (1999). *Systems Thinking, Systems Practice. Soft Systems Methodology: A 30-year Retrospective*. Chichester: John Wiley & Sons, Ltd.
- 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(4), 650-655.
- Cockburn, A. (1998). *Surviving Object-oriented Projects: A Manager's Guide*. Reading MA: Addison Wesley Longman.
- Cockburn, A. (2002). *Agile Software Development*. Boston: Addison-Wesley.
- Conboy, K. (2004). Agile methods: The gap between theory and practice. In J. Eckstein & H. Baumeister (Eds.), *Extreme Programming and Agile Processes in Software Engineering. Lecture Notes in Computer Science. LNCS 3092* (Vol. 316, pp. 316). Heidelberg Berlin: Springer-Verlag.
- Conboy, K., & Fitzgerald, B. (2007a). Agile drivers, capabilities, and value: An overarching assessment framework for systems development. In K. Desouza, C (Ed.), *Agile Information Systems: Conceptualization, Construction, and Management* (pp. 207-221). Amsterdam: Elsevier.
- Conboy, K., & Fitzgerald, B. (2007b). The views of experts on the current state of agile method tailoring. In T. McMaster, D. Wastell, E. Ferneley & J. DeGross (Eds.), *IFIP International Federation for Information Processing* (Vol. 235, pp. 217-234). Boston: Springer.
- Conboy, K., Fitzgerald, B., & Golden, W. (2005). Agility in information systems development: A three-tiered framework. In R. Baskerville, L. Mathiassen, J. Pries-Heje & J. DeGross (Eds.), *Business Agility and Information Technology Diffusion. IFIP TC8 WG 8.6 International Working Conference May 8-11, 2005, Atlanta, Georgia, U.S.A.* (pp. 36-49). New York: Springer.
- Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28-31.
- Coplien, J., Hutz, S., & Marykuca, B. (1992). Iterative development/OO: The bottom line. *OOPS Messenger*, 4(2), 101-108.
- Creswell, J. W. (2003). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (2 ed.). Thousand Oaks, CA: Sage.
- Cronholm, S., & Agerfalk, P. J. (1999). On the concept of method in information systems development. Proceedings of the 22nd Information Systems (Vol. 4, pp. 229-236): Linkoping. Retrieved from <http://www.vits.org/publikationer/dokument/52.pdf>.
- Crowston, K. (1991). *Towards a coordination cookbook: recipes for multi-agent action*. PhD, Massachusetts Institute of Technology, Cambridge, Mass. Retrieved from <http://hdl.handle.net/1721.1/49303>
- Crowston, K. (1994). A taxonomy of organizational dependencies and coordination mechanisms. Retrieved from <http://ccs.mit.edu/papers/CCSWP174.html>

- Crowston, K. (2003). A taxonomy of organizational dependencies and coordination mechanisms. In T. W. Malone, K. Crowston & G. A. Herman (Eds.), *Organizing Business Knowledge: The MIT Process Handbook* (pp. 86-108). Cambridge, Massachusetts: The MIT Press.
- Crowston, K., Hala, A., Howison, J., & Chengetai, M. (2004). Effective work practices for software engineering: free/libre open source software development. *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research WISER 04* (pp. 18-26). New York, NY, USA: ACM.
- Crowston, K., & Kammerer, E. E. (1998). Coordination and collective mind in software requirements development. *IBM Systems Journal* 37(2).
- Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., & Howison, J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology*, 49(6), 564-575.
- Crowston, K., & Osborn, C. S. (2003). A coordination theory approach to process description and redesign. In T. W. Malone, K. Crowston & G. A. Herman (Eds.), *Organizing Business Knowledge: The MIT Process Handbook* (pp. 335-370). Cambridge, Massachusetts: The MIT Press.
- Crowston, K., Rubleske, J., & Howison, J. (2006). Coordination theory: a ten-year retrospective. In P. Zhang & D. Galletta (Eds.), *Human-Computer Interaction and Management Information Systems: Foundations* (pp. 120-138). Armonk, New York: Sharpe, M E.
- Cummings, J. N., Espinosa, A. J., & Pickering, C. K. (2009). Crossing spatial and temporal boundaries in globally distributed projects: A relational model of coordination delay. *Information Systems Research*, 20(3), 420-439.
- Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, 31(11), 1268-1287.
- Cusumano, M., & Selby, R. W. (1997). How Microsoft builds software. *Communications of the ACM*, 40(6), 53-61.
- Cusumano, M., & Yoffie, D. B. (1999). Software development on Internet time. *IEEE Computer*, 32(10), 60-69.
- Darke, P., Shanks, G., & Broadbent, M. (1998). Successfully completing case study research: Combining rigour, relevance and pragmatism. *Information Systems Journal*, 8(4), 273-289.
- Deephouse, C., Mukhopadhyay, T., Goldenson, D. R., & Kellner, M. I. (Winter 1995-96). Software processes and project performance. *Journal of Management Information Systems*, 12(3), 187-205.
- DeLone, W. H., & McLean, E. R. (1992). Information systems success: The quest for the dependent variable. *Information Systems Research*, 3(1), 60-95.
- DeLone, W. H., & McLean, E. R. (2003). The DeLone and McLean model of information systems success: A ten year update. *Journal of Management Information Systems*, 19(4), 9-30.

- DeSanctis, G., & Jackson, M. B. (1994). Coordination of information technology management: Team based structures and computer-based communication system. *Journal of Management Information Systems*, 10(4), 85-110.
- Dingosyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *The Journal of Systems and Software*, 85 (6), 1213-1221.
- Dingsoyr, T., Dyba, T., & Abrahamsson, P. (2008). A preliminary roadmap for empirical research on agile software development. *Proceedings of the Agile 2008 Conference* (pp. 83-94): IEEE Xplore digital library.
- Domino, M., Hevner, A., & Collins, R. W. (2002). Applying agile software development processes to global virtual teams: A study of communication modalities. *Proceedings of the Annual ACM Special Interest Group on Computer Personnel Research Conference, SIGCPR 2002* (pp. 76-78). New York, USA: ACM Press.
- Doty, D. H., & Glick, W. H. (1994). Typologies as a unique form of theory building: Toward improved understanding and modeling. *The Academy of Management Review*, 19(2), 230-251.
- Dove, R. (2005). Agile enterprise cornerstones: Knowledge, values, and response ability. In R. Baskerville, L. Mathiassen, J. Pries-Heje & J. DeGross (Eds.), *Business Agility and Information Technology Diffusion IFIP TC8 WG 8.6 International Working Conference, May 8-11, Atlanta, Georgia, U.S.A.* (pp. 313-330). New York: Springer.
- Dube, L., & Pare, G. (2003). Rigor in information systems positivist case research: Current practice, trends, and recommendations. *MIS Quarterly*, 27(4), 597-635.
- Dubin, R. (1978). *Theory building*. New York: The Free Press.
- Dvir, D., Lipovetsky, S., Shenhar, A., & Tishler, A. (1998). In search of project classification: A non-universal approach to project success factors. *Research Policy*, 27(9), 915-935.
- Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10), 833-859.
- Dyba, T., Moe, N., & Arisholm, E. (2005). Measuring software methodology use: Challenges of conceptualization and operationalization. *International Symposium on Empirical Software Engineering, 2005* (pp. 447-457): IEEE Xplore digital library.
- Dynamic Systems Development Method, Version 2*. (1995). (2 ed.). Ashford: Tesseract Publishing.
- Edmondson, A. C., & McManus, S. E. (2007). Methodological fit in management field research. *Academy of Management Review*, 32(4), 1155-1179.
- Eisenhardt, K. M. (1989). Building theories from case study research. *The Academy of Management Review*, 14(4), 532-550.
- Eisenhardt, K. M., & Graebner, M. E. (2007). Theory building from cases: Opportunities and challenges. *Academy of Management Journal*, 50(1), 25-32.

- Eisenhardt, K. M., & Martin, J. (2000). Dynamic capabilities: What are they? *Strategic Management*, 21(10-11), 1105-1121.
- Elshamy, A., & Elssamadisy, A. (2007). Applying agile to large projects: new agile software development practices for large projects. In G. Concas, E. Damiani, M. Scotto & G. Succi (Eds.), *Agile Processes in Software Engineering and Extreme Programming. Proceedings of the 8th International XP Conference, XP 2007, Como, Italy, June 18-22* (Vol. 4536, pp. 46-53). Heidelberg, Berlin: Springer.
- 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 E. Salas & S. M. Fiore (Eds.), *Team Cognition: Understanding the Factors that Drive Process and Performance* (pp. 107-129). Washington DC: American Psychological Association.
- Espinosa, A. J., Lerch, J., & Kraut, R. (2002). Explicit vs. implicit coordination mechanisms and task dependencies: One size does not fit all. Retrieved from <http://www.cs.cmu.edu/afs/cs.cmu.edu/user/kraut/www/RKraut.site.files/articles/Espinosa03-ExplicitVsImplicitCoordination.pdf>
- Espinosa, A. J., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007). Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, 24(1), 135-169.
- Eva, M. (1994). *SSADM Version 4: A User's Guide* (2 ed.). London: McGraw-Hill Book Company.
- Faraj, S., & Sambamurthy, V. (2006). Leadership of information systems development projects. *IEEE Transactions on Engineering Management*, 53(2), 238-249.
- Faraj, S., & Sproull, L. (2000). Coordinating expertise in software development teams. *Management Science*, 46(12), 1554-1568.
- Feldman, M. S., & Orlikowski, W. J. (2011). Theorizing practice and practicing theory. *Organization Science*, 22(5), 1240-1253.
- Fiore, S. M., & Salas, E. (2004). Why we need team cognition. In E. Salas & S. M. Fiore (Eds.), *Team Cognition* (pp. 235-248). Washington, DC: American Psychological Association.
- Fitzgerald, B. (1997). The use of systems development methodologies in practice: A field study. *Information Systems Journal*, 7(3), 201-212.
- Fitzgerald, B. (2000). Systems development methodologies: The problem of tenses. *Information Technology and People*, 13(3), 174-185.
- Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems*, 15(2), 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(4), 39-68.

- Fruhling, A., Tyser, K., & de Vreede, G. J. (2005). Experiences with Extreme Programming in telehealth: Developing and implementing a biosecurity health care application *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS'05* (pp. 1-10): IEEE Xplore digital library.
- Galbraith, J. R. (1973). *Designing complex organizations*. Reading, MA: Addison-Wesley.
- Galbraith, J. R. (1974). Organization design: an information processing view. *Interfaces*, 4(3), 28-36.
- Galbraith, J. R. (1977). *Organization design*. Reading, MA: Addison-Wesley.
- Gallivan, M. J. (1997). Value in triangulation: a comparison of two approaches for combining qualitative and quantitative methods. In A. S. Lee, J. Liebenau & J. I. DeGross (Eds.), *Proceedings of the IFIP TC8 WG 8.2 International Conference on Information Systems and Qualitative Research, 31 May-3rd June 1997, Philadelphia, Pennsylvania, USA* (pp. 417-443). London: Chapman & Hall.
- Gilb, T. (2005). *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software*. Oxford: Elsevier Butterworth-Heinemann.
- Gittell, J. H. (2011). Relational coordination: guidelines for theory, measurement and analysis. 1-92. Retrieved from http://rcrc.brandeis.edu/downloads/Relational_Coordination_Guidelines_8-25-11.pdf
- Gittell, J. H., Seidner, R., & Wimbush, J. (2010). A relational model of how high-performance work systems work. *Organization Science*, 21(2), 490-506. doi: 10.1287/orsc.1090.0446
- Glaser, B., & Strauss, A. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. London: Wiedenfeld and Nicholson.
- Glass, R., L., Ramesh, V., & Vessey, I. (2004). An analysis of research in computing disciplines. *Communications of the ACM*, 47(6), 89-94.
- Goulielmos, M. (2004). Systems development approach: Transcending methodology. *Information Systems Journal*, 14(4), 363-386.
- Graham, I., Henderson-Sellers, B., & Younessi, H. (1997). *The OPEN Process Specification*. Harlow, England: Addison-Wesley.
- Gregor, S. (2006). The nature of theory in information systems. *MIS Quarterly*, 30(3), 611-642.
- Grinter, R. E. (1996). *Understanding dependencies: A study of the coordination challenges in software development*. PhD, University of California, Irvine.
- Grossman, F., Bergin, J., Leip, D., Merrit, S., & Gotel, O. (2004). One XP experience: Introducing agile (XP) software development into a culture that is willing but not ready. In H. Lutfiyya, J. Singer & D. A. Stewart (Eds.), *Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research CASCOS '04* (pp. 242-254): IBM Press.

- Gunter, M., Gisler, M., & von Bredow, B. (2002). New software development paradigms and possible adoption for security. *Proceedings of the 36th International Conference on Systems Sciences HICSS'03*: IEEE Xplore digital library.
- Hansson, C., Dittrich, Y., Gustafsson, B., & Zarnak, S. (2006). How agile are industrial software development practices? *The Journal of Systems and Software*, 79(9), 1295-1311.
- Harris, M., Aebischer, K., & Klaus, T. (2007). The whitewater process: software product development in small IT businesses. *Communications of the ACM*, 50(5), 89-93.
- Harris, M., Collins, R. W., & Hevner, A. R. (2009). Control of flexible software development under uncertainty. *Information Systems Research*, 20(3), 400-419.
- Harris, M., Hevner, A. R., & Collins, R. W. (2006). Controls in flexible software development. *Proceedings of the 39th Hawaii International Conference on System Sciences, HICSS'06*: IEEE Xplore digital library.
- Hatch, M. J., & Cunliffe, A. L. (2006). *Organization Theory* (2 ed.). Oxford: Oxford University Press.
- Henderson-Sellers, B. (2003). Method engineering for OO systems development. *Communications of the ACM*, 46(10), 73-78.
- Herbsleb, J. D. (2007). Global Software Engineering: The future of socio-technical coordination. *Proceedings of the International Conference on Software Engineering: Future of Software Engineering FOSE '07* (pp. 188-198): IEEE Xplore digital library.
- Herbsleb, J. D., & Grinter, R. E. (1999). Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 16(5), 63-70.
- Herbsleb, J. D., & Mockus, A. (2003). Formulation and preliminary test of an empirical theory of coordination in software engineering *Proceedings of the 9th European Software Engineering Conference* (pp. 138-147). New York, NY: ACM.
- Herman, G. A., & Malone, T. W. (2003). What is in the Process Handbook. In T. W. Malone, K. Crowston & G. A. Herman (Eds.), *Organizing Business Knowledge: the MIT Process Handbook* (pp. 221-258). Cambridge, Massachusetts: The MIT Press.
- Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, NY: Dorset House Publishing.
- Highsmith, J. (2002). *Agile Software Development Ecosystems*. Boston: Addison-Wesley.
- 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(4), 41-61.
- Hirschheim, R., & Klein, H. K. (1989). Four paradigms of information systems development. *Communications of the ACM*, 32(10), 1199-1216.

- Hirschheim, R., Klein, H. K., & Lyytinen, K. (1996). Exploring the intellectual structures of information systems development: as social theoretic analysis. *Accounting, Management, and Information Technology*, 6(1/2), 1-64.
- Hoda, R., Noble, J., & Marshall, S. (2010). Organizing self-organizing teams. *Proceedings of the IEEE/ACM International Conference on Software Engineering, ICSE 2010, May 2-9, 2010* (pp. 285-294). New York, NY: ACM.
- Hoegl, M., & Gemuenden, H. G. (2001). Teamwork quality and the success of innovative products. *Organization Science*, 12(4), 435-449.
- Hong, W., Thong, J. Y. L., Chasalow, L., & Dhillon, G. (2011). User acceptance of agile information systems: a model and empirical test. *Journal of Management Information Systems*, 28(1), 235-272.
- Hsu, J. S.-C., Shih, S.-P., Chiang, J. C., & Liu, J. Y.-C. (2012). The impact of transactive memory systems in IS development teams' coordination, communication, and performance. *International Journal of Project Management*, 30(3), 329-340.
- Hunt, A., & Thomas, D. (2000). *The Pragmatic Programmer*. Boston: Addison Wesley.
- Iivari, J., Hirschheim, R., & Klein, H. K. (2001). A dynamic framework for classifying information systems development methodologies and approaches. *Journal of Management Information Systems*, 17(3), 179-218.
- 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(4), 313-342.
- Iivari, J., & Iivari, N. (2011). The relationship between organizational culture and the deployment of agile methods. *Information and Software Technology*, 53(5), 509-520. doi: 10.1016/j.infsof.2010.10.008
- Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Reading, Massachusetts: Addison-Wesley.
- Jayarathna, N. (1994). *Understanding and Evaluating Methodologies NIMSAD: A Systematic Framework*. London: McGraw-Hill.
- Jugdev, K., & Muller, R. (2005). A retrospective look at our evolving understanding of project success. *Project management journal*, 36(4), 19-31.
- Kahkonen, T., & Abrahamsson, P. (2003). Digging into the fundamentals of extreme programming: Building the theoretical base for agile methods. *Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture"* (pp. 273-280): IEEE Xplore digital library.
- 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(12), 1681-1710.
- Kaplan, A. (1998). *The Conduct of Inquiry: Methodology for Behavioural Science*. New Brunswick, NJ: Transaction Publishers.

- Kautz, K., Hansen, B., & Jacobsen, D. (2004). The utilization of information systems development methodologies in practice. *Journal of Information Technology Cases and Applications*, 6(4), 1-20.
- Kautz, K., Madsen, S., & Norbjerg, J. (2007). Continuing the debate: A response to a response - persistent problems and practices in information systems development as enduring contradictions of new software development approaches? *Information Systems Journal*, 17(3), 247-249.
- Kautz, K., & Zumpe, S. (2008). Just enough structure at the edge of chaos: Agile information system development in practice. In P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan & X. Wang (Eds.), *Proceedings of the 9th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP 2008, Limerick Ireland, June 10-14, 2008* (Vol. LNBIP 9, pp. 137-146). Berlin, Heidelberg: Springer.
- Keil, M. (1995). Pulling the plug: Software project management and the problem of project escalation. *MIS Quarterly*, 19(4), 421-447.
- Kerzner, H. (2003). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling* (8 ed.). Hoboken, New Jersey: John Wiley & Sons.
- Kirsch, L. J. (1996). The management of complex tasks in organizations: Controlling the systems development process. *Organization Science*, 7(1), 1-21.
- Kirsch, L. J. (1997). Portfolios of control modes in IS project management. *Information Systems Research*, 8(3), 214-239.
- 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(2), 96-108.
- Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM*, 38(3), 69-81.
- Kruchten, P. (2000). *The Rational Unified Process: An Introduction* (2 ed.). Boston: Addison-Wesley Longman.
- Kruchten, P. (2008). What do software architects really do? *The Journal of Systems and Software*, 81(12), 2413-2416.
- Kuhn, S., & Muller, M. J. (1993). Participatory design. *Communications of the ACM*, 36(4), 25-28.
- Kumar, K., & Welke, R. J. (1992). Methodology engineering: A proposal for situation-specific methodology construction. In W. W. Cotterman & J. A. Senn (Eds.), *Challenges and Strategies for Research in Systems Development* (pp. 257-269). New York: John Wiley & Sons.
- Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, 36(6), 47-56.
- Lawrence, P. R., & Lorsch, J. W. (1967). Differentiation and integration in complex organisations. *Administrative Science Quarterly*, 12(1), 1-47.

- Lee, A. S. (1989). A scientific methodology for MIS case studies. *MIS Quarterly*, 13(1), 33-50.
- Lee, A. S. (1991). Integrating positivist and interpretive approaches to organizational research. *Organization Science*, 2(4), 342-365.
- Levina, N., & Vaast, E. (2005). The emergence of boundary spanning competence in practice: Implications for implementation and use of information systems. *MIS Quarterly*, 29(5), 335-363.
- Levitt, R. E. (2011). Toward project management 2.0. In M. T. Tool (Ed.), *Proceedings of the Engineering Project Organizations Conference*. Estes Park, Colorado: EPOS.
- Levitt, R. E., Thomsen, J., Christiansen, T. R., Kunz, J. C., Jin, Y., & Nass, C. (1999). Simulating project work processes and organizations: Toward a micro-contingency theory of organizational design. *Management Science*, 45(11), 1479-1495.
- Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., . . . Kahkonen, T. (2004). Agile software development in large organizations. *Computer*, 37(12), 26-33.
- Little, T. (2005). Context-adaptive agility: Managing complexity and uncertainty. *IEEE Software*, 22(3), 28-35.
- Liu, F., & Myers, M. D. (2011). An analysis of the AIS basket of top journals. *Journal of Systems and Information Technology*, 13(1), 5-24.
- Luftman, J., & McLean, E. R. (2004). Key issues for IT executive. *MIS Quarterly Executive*, 3(2), 89-104.
- Lyytinen, K., & Newman, M. (2008). Explaining information systems change: A punctuated socio-technical change model. *European Journal of Information Systems*, 17(1), 589-613.
- Lyytinen, K., & Rose, G. M. (2004). Explaining radical innovation in system development organizations. *Sprouts: Working Papers in Information Environments, Systems and Organizations*, 4(1). Retrieved from <http://sprouts.aisnet.org/4-1>
- Lyytinen, K., & Rose, G. M. (2005). How agile is agile enough? Toward a theory of agility in software development. In R. Baskerville, L. Mathiassen, J. Pries-Heje & J. DeGross (Eds.), *Business Agility and Information Technology Diffusion, IFIP TC8 WG 8.6 International Working Conference, May 8-11, Atlanta, Georgia, U.S.A.* (pp. 203-225). New York: Springer.
- Lyytinen, K., & Rose, G. M. (2003a). Disruptive information system innovation: The case of internet computing. *Information Systems Journal*, 13(4), 301-330.
- Lyytinen, K., & Rose, G. M. (2003b). The disruptive nature of information technology innovations: The case of internet computing in systems development organisations. *MIS Quarterly*, 27(4), 557-595.
- MacCormack, A., Rusnak, J., & Baldwin, C. Y. (2008). Exploring the duality between product and organizational architectures: a test of the mirroring hypothesis.

Harvard Business School Technology & Operations Mgt. Unit Research Paper No. 08-039. Retrieved from
http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1104745

- MacCormack, A., Verganti, R., & Iansiti, M. (2001). Developing products on "Internet Time": The anatomy of a flexible development process. *Management Science*, 47(1), 133-150.
- MacKenzie, A., & Monk, S. (2004). From cards to code: How extreme programming re-embodies programming as collective practice. *Computer Supported Cooperative Work*, 13(1), 91-117.
- Madsen, S. (2007). Conceptualising the causes and consequences of uncertainty in IS development organisations and projects. In H. Osterle, J. Schelp & R. Winter (Eds.), *Proceedings of the 15th European Conference on Information Systems, ECIS 2007, St. Gallen, Switzerland* (pp. 855-864). Retrieved from
<http://aisel.aisnet.org/ecis2007/156>.
- Malone, T. W. (1987). Modeling coordination in organizations and markets. *Management Science*, 33(10), 1317-1332.
- Malone, T. W. (1988). What is coordination theory? *Sloane School of Management Working Paper # 2051-88*. Retrieved from <http://hdl.handle.net/1721.1/2208>
- Malone, T. W. (1989). Center for Coordination Science Massachusetts Institute of Technology. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems CHI'89* (pp. 145-146). New York, NY: ACM.
- Malone, T. W., & Crowston, K. (1990). What is coordination theory and how can it help design cooperative work systems? *Proceedings of the ACM Conference on Computer-supported Cooperative Work* (pp. 357-370). New York, NY: ACM.
- Malone, T. W., & Crowston, K. (1991). Toward an Interdisciplinary Theory of Coordination. Retrieved from <http://hdl.handle.net/1721.1/2356>
- Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1), 87-119.
- Malone, T. W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., . . . O'Donnell, E. (1999). Tools for inventing organizations: toward a handbook of organizational processes. *Management Science*, 45(3), 425-443.
- March, J. G. (1991). Exploration and exploitation in organizational learning. *Organization Science*, 2(1), 71-87.
- March, J. G., & Simon, H. A. (1958). *Organization*. New York: Wiley.
- March, J. G., & Sutton, R. I. (1997). Organizational performance as a dependent variable. *Organization Science*, 8(6), 698-706.
- Markus, M. L., & Robey, D. (1988). Information technology and organizational change: causal structure in theory and research. *Management Science*, 34(5), 583-598.
- Martin, J. (1989). *Information Engineering Book 1 Introduction*. Upper Saddle River, New Jersey: Prentice Hall PTR.

- Martin, J. (1991). *Rapid Application Development*. Englewood Cliffs, NJ: Prentice Hall.
- Martin, J., & Finkelstein, C. (1981). *Information Engineering. Vol 1 and 2*. Englewood Cliffs, New Jersey: Prentice Hall.
- Maruping, L. M., Venkatesh, V., & Agarwal, R. (2009). A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research*, 20(3), 277-399.
- Mathiassen, L., & Pedersen, K. (2008). Managing uncertainty in organic development projects. *Communications of the Association for Information Systems*, 23(27), 483-500.
- Matook, S., & Kautz, K. (2008). Mindfulness and agile software development. In S. L. Huff (Ed.), *Proceedings of the 19th Australasian Conference on Information Systems, ACIS 2008* (pp. 638-646). Retrieved from <http://aisel.aisnet.org/acis2008/58>.
- Maurer, F. (2002). Supporting distributed extreme programming *Proceedings of the Extreme Programming and Agile Methods, XP/Agile Universe 2002: Second XP Universe and First Agile Universe Conference, August 4-7* (Vol. 2418 Lecture Notes in Computer Science, pp. 13-22). Berlin: Springer-Verlag.
- McBride, T. (2008). The mechanisms of project management of software development. *The Journal of Systems and Software*, 81(12), 2386-2395.
- McChesney, I. R. (1997). Effective coordination in the software process - historical perspectives and future directions. *Software Quality Journal*, 6(3), 235-246.
- McChesney, I. R., & Gallagher, S. (2004). Communication and coordination practices in software engineering projects. *Information and Software Technology*, 46(7), 473-489.
- Mead, S. P. (2001). Using social network analysis to visualize project teams. *Project Management Journal*, 31(4), 32-38.
- Meso, P., & Jain, R. (2006). Agile software development: Adaptive systems principles and best practices. *Information Systems Management*, 23(3), 19-30.
- Miles, M. B., & Huberman, A. M. (1994). *Qualitative Data Analysis* (2 ed.). Thousand Oaks: Sage.
- Mintzberg, H. (1980). Structure in 5's: A synthesis of the research on organization design. *Management Science*, 26(3), 322-341.
- . The MIT Process Handbook Project. (2003) Retrieved 1 May, 2009, from <http://ccs.mit.edu/ph/>
- Moe, N., Dingsoyr, T., & Dyba, T. (2008). Understanding self-organizing teams in agile software development. *Proceedings of the 19th Australian Conference on Software Engineering, ASWEC 2008* (pp. 76-85): IEEE Xplore digital library.
- 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(5), 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(4), 876-910.
- Morrison, J., & George, J. F. (1995). Exploring the software engineering component of MIS research. *Communications of the ACM*, 38(7), 80-91.
- Nandhakumar, J., & Avison, D. (1999). The fiction of methodological development: A field study of information systems development. *Information Technology and People*, 12(2), 176-191.
- Napier, N., Keil, M., & Tan, F. B. (2009). IT project managers' construction of successful project management practice: A repertory grid investigation. *Information Systems Journal*, 19(3), 255-282.
- Naur, P., & Randell, B. (1968). *Software Engineering: Report of a conference sponsored by the NATO science committee, 7-11 October 1968, Garmisch, Germany*. Retrieved from <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>
- Nawrocki, J., Jasinski, M., Walter, B., & Wojciechowski, A. (2002). Extreme programming modified: Embrace requirements engineering practices. *Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE'02* (pp. 303-310): IEEE Xplore digital library.
- Nelson, R. R. (2007). IT Project management: Infamous failures, classic mistakes, and best practices. *MIS Quarterly Executive*, 6(2), 67-78.
- Nerur, S., & Balijepally, V. (2007). Theoretical reflections on agile development methodologies. *Communications of the ACM*, 50(3), 79-83.
- Neufeld, D., Fang, Y., & Huff, S. (2007). The IS identity crisis. *Communications of the Association for Information Systems*, 19(19), 447-464. Retrieved from <http://aisel.aisnet.org/cais/vol19/iss1/19>
- Nidumolu, S. (1995). The effect of coordination and uncertainty on software project performance: Residual performance risk as an intervening variable. *Information Systems Research*, 6(3), 191-219.
- Nidumolu, S. (1996). A comparison of the structural contingency and risk-based perspectives on coordination in software-development projects. *Journal of Management Information Systems*, 13(2), 77-113.
- Nidumolu, S., & Subramani, M., R. (2004). The matrix of control: Combining process and structure approaches to managing software development. *Journal of Management Information Systems*, 20(3), 159-196.
- Nolan, R. L., & Wetherbe, J. C. (1980). Toward a comprehensive framework for MIS research. *MIS Quarterly*, 4(2), 1-19.
- Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science*, 5(1), 14-37.
- Nord, R. L., & Tomayko, J. E. (2006). Software architecture-centric methods and agile development. *IEEE Software*, March/April 2006, 47-53.

- Ohno, T. (1988). *Toyota Production System: Beyond Large-scale Production*. Tokyo: Diamond Inc.
- Okhuysen, G. A., & Bechky, B. A. (2009). Coordination in organizations: an integrative perspective. *The Academy of Management Annals*, 3(1), 463-502.
- Olle, T. W., Sol, H. C., & Tully, C. (Eds.). (1983). *Information Systems Design Methodologies: A Feature Analysis. Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies*. Amsterdam: North-Holland.
- Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research*, 2(1), 1-28.
- Ouchi, W. G. (1977). The relationship between organizational structure and organizational control. *Administrative Science Quarterly*, 22(1), 95-113.
- Ouchi, W. G. (1979). A conceptual framework for the design of organisational control mechanisms. *Management Science*, 25(9), 833-848.
- Ouchi, W. G. (1980). Markets, bureaucracies & clans. *Administrative Science Quarterly*, 25(1).
- Ovaska, P., Rossi, M., & Marttiin, P. (2003). Architecture as a coordination tool in multi-site software development. *Software Process Improvement and Practice*, 8(4), 233-247. doi: 10.1002/spip.186
- Paasivaara, M., & Lassenius, C. (2004). Using iterative and incremental processes in global software development. *Proceedings of the Third International Workshop on Global Software Development (GSD 2004), Edinburgh, Scotland, UK* (pp. 45-47).
- Palmer, S. R., & Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development*. Upper Saddle River: Prentice Hall.
- Pare, G. (2004). Investigating information systems with positivist case study research. *Communications of the Association for Information Systems*, 13(1), 233-264.
- Pare, G., & Elam, J. J. (1997). Using case study research to build theories of IT implementation. In A. S. Lee, J. Liebenau & J. I. DeGross (Eds.), *Proceedings of the IFIP TC8 WG 8.2 International Conference on Information Systems and Qualitative Research, 31 May-3rd June 1997, Philadelphia, Pennsylvania, USA* (pp. 542-568). London: Chapman & Hall.
- Parnas, D. L., & Clements, P. C. (1986). A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12(2), 251-257.
- Parolia, N., Goodman, S., Li, Y., & Jiang, J. J. (2007). Mediators between coordination and IS project performance. *Information and Management*, 44(7), 635-645.
- Patel, N., V (Ed.). (2003). *Adaptive Evolutionary Information Systems*. Hershey: Idea Group Publishing.

- Petter, S., DeLone, W. H., & McLean, E. R. (2008). Measuring information systems success: Models, dimensions, measures, and interrelationships. *European Journal of Information Systems*, 17, 236-263. doi: 10.1057/ejis.2008.15
- 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(3), 303-337. doi: 10.1007/s10664-008-9065-9
- Poppendiek, M., & Poppendiek, T. (2003). *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley.
- Pries-Heje, L., & Pries-Heje, J. (2011). Why Scrum works. Proceedings of the Agile Conference 2011, Salt Lake City, UT (pp. 20-28): IEEE Xplore digital library. doi: 10.1109/AGILE.2011.34
- Rajlich, V. (2006). Changing the paradigm of software engineering. *Communications of the ACM*, 49(8), 67-70.
- Ramesh, B., Abrahamsson, P., Cockburn, A., Lyytinen, K., & Williams, L. (2005). Agile software development methods: When and why do they work? In R. Baskerville, L. Mathiassen, J. Pries-Heje & J. DeGross (Eds.), *Proceedings of the Business Agility and Information Technology Diffusion IFIP TC8 WG 8.6 International Working Conference May 8-11, Atlanta, Georgia* (pp. 371-374). Berlin: Springer.
- Ramesh, B., Cao, L., & Baskerville, R. (2007). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal*, 20(5), 449-480. doi: 10.1111/j.1365-2575.2007.00259.x
- Ramesh, B., Pries-Heje, J., & Baskerville, R. (2002). Internet software engineering: A different class of processes. *Annals of Software Engineering*, 14(1-4), 169-195. doi: 10.1023/A:1020557725165
- Ratbe, D., King, W. R., & Kim, Y.-G. (2000). The fit between project characteristics and application development methodologies: A contingency approach. *The Journal of Computer Information Systems*, 40(2), 26-33.
- Reeves, M., & Zhu, J. (2004). Moomba - a collaborative environment for supporting distributed extreme programming in global software development. In J. Eckstein & H. Baumeister (Eds.), *Proceedings of the Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2004* (Vol. 3092 Lecture Notes in Computer Science, pp. 38-50). Berlin: Springer-Verlag.
- Reifer, D. J., Maurer, F., & Erdogmus, H. (2003). Scaling agile methods. *IEEE Software*, 20(4), 12-14.
- Ren, Y., Kiesler, S., & Fussell, S. (2008). Multiple group coordination in complex and dynamic task environments: Interruptions, coping mechanisms, and technology recommendations. *Journal of Management Information Systems*, 25(1), 105-130.
- Rico, R., Sanchez-Manzanares, M., Gil, F., & Gibson, C. (2008). Team implicit coordination processes: A team knowledge-based approach. *Academy of Management Review*, 33(1), 163-184.
- Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE Software*, 17(4), 26-32.

- Robinson, H., & Sharp, H. (2004). The characteristics of XP teams. In J. Eckstein & H. Baumeister (Eds.), *Proceedings of Extreme Programming and Agile Processes in Software Engineering, XP 2004* (Vol. LNCS 3092, pp. 139-147). Berlin: Springer-Verlag.
- Rodriguez, D., Sicilia, M. A., E, G., & Harrison, R. (2012). Empirical findings on team size and productivity in software development. *The Journal of Systems and Software*, 85(3), 562-570.
- Roseman, M., & Vessey, I. (2008). Toward improving the relevance of information systems research to practice: The role of applicability checks. *MIS Quarterly*, 32(1), 1-22.
- Royce, W. W. (1987). Managing the development of large software systems *Proceedings of the 9th International Conference on Software Engineering (Reprinted from Proceedings, IEEE WESCON, August 1970 p. 1-9. Originally published by TRW)* (pp. 328-338). Los Alamitos, CA: IEEE Computer Society Press
- Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Massachusetts: Addison-Wesley.
- Russo, N. L., & Stolterman, E. (2000). Exploring the assumptions underlying information systems methodologies: Their impact on past, present and future ISM research. *Information Technology and People*, 13(4), 313.
- Sabherwal, R. (2003). The evolution of coordination in outsourced software development projects: A comparison of client and vendor perspectives. *Information and Organization*, 13(3), 153-202.
- Sabherwal, R., & Choudhury, V. (2006). Governance of remotely outsourced software development: A comparison of client and vendor perspectives. In R. Hirschheim, A. Heinzl & J. Dibbern (Eds.), *Information Systems Outsourcing* (2 ed.). Berlin: Springer.
- Salo, O., Kolehmainen, K., Kyllonen, P., Lothman, J., Salmijarvi, S., & Abrahamsson, P. (2004). Self-adaptability of agile software processes: A case study on post-iteration workshops. In J. Eckstein & H. Baumeister (Eds.), *5th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2004, Garmisch-Partenkirchen, Germany* (Vol. LNCS 3092, pp. 184-193). Heidelberg: Springer-Verlag.
- Sambamurthy, V., & Kirsch, L. J. (2000). An integrative framework of the information systems development process. *Decision Sciences*, 31(2), 391-411.
- Schalliol, G. (2003). Challenges for analysts on a large XP project. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme Programming Perspectives* (pp. 375-385). Boston: Addison-Wesley.
- Schummer, T., & Schummer, J. (2001). Support for distributed teams in extreme programming. In G. Succi & M. Marchesi (Eds.), *Extreme Programming Examined* (pp. 355-377). Boston: Addison-Wesley Longman.
- Schwaber, K. (1995). Scrum software development process. *Proceedings of OOPSLA 1995*. Retrieved from <http://home.hib.no/ai/data/master/mod251/2009/articles/scrum.pdf>

- Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum*. Upper Saddle River, New Jersey: Prentice Hall.
- Schwalbe, K. (2010). *Information Technology Project Management* (6 ed.). Australia: Thomson Course Technology.
- Seaman, C. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4), 557-572.
- Shapira, Z. (2011). "I've got a theory paper - do you?": Conceptual, empirical, and theoretical contributions to knowledge in the organizational sciences. *Organization Science*, 22(5), 1312-1321.
- Sharp, H., & Robinson, H. (2004). An ethnographic study of XP practice. *Empirical Software Engineering*, 9(4), 353-375.
- Sharp, H., & Robinson, H. (2008). Collaboration and co-ordination in mature eXtreme programming teams. *International Journal of Human-Computer Studies*, 66(7), 506-518.
- Sidorova, A., Evangelopoulos, N., Valacich, J. S., & Ramakrishnan, T. (2008). Uncovering the intellectual core of the information systems discipline. *MIS Quarterly*, 32(3), 467-482.
- Spradley, J. P. (1979). *The Ethnographic Interview*. New York: Holt, Rinehart and Winston.
- Stapleton, J. (1997). *DSDM Dynamic Systems Development Method*. Harlow, England: Addison-Wesley.
- Stephens, M., & Rosenberg, D. (2003). *Extreme Programming Refactored: The Case Against XP*. Berlin: Springer.
- Straub, D., Boudreau, M.-C., & Gefen, D. (2004). Validation guidelines for IS postivist research. *Communications of the Association for Information Systems*, 13, Article 24, 380-427. Retrieved from <http://aisel.aisnet.org/cais/vol13/iss1/24>
- Straub, D., Gefen, D., & Boudreau, M.-C. (2005). Quantitative research. In D. Avison & J. Pries-Heje (Eds.), *Research in information systems: a handbook for research supervisors and their students* (pp. 221-238). Amsterdam: Elsevier Butterworth Heinemann.
- Strode, D. E. (2005). *The agile methods: An analytical comparison of five agile methods and an investigation of their target environment*. Master of Information Sciences in Information Systems Thesis, Massey University, Palmerston North, New Zealand. Retrieved from <http://hdl.handle.net/10179/515>
- Strode, D. E. (2006). Agile methods: a comparative analysis. In S. Mann & N. Bridgeman (Eds.), *Proceedings of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications, NACCQ 2006* (pp. 257-264). Wellington: NACCQ.
- Strode, D. E. (2007). Characterising the agile methods. *New Zealand Journal of Applied Computing and Information Technology*, 11(1), 65-79.

- Strode, D. E., Hope, B., Huff, S. L., & Link, S. (2011). Coordination effectiveness in an agile software development context. In P. B. Seddon & S. Gregor (Eds.), *Proceedings of the Pacific Asia Conference on Information Systems, PACIS 2011, July 2011, Brisbane, Australia* (pp. 1-16). Retrieved from <http://aisel.aisnet.org/pacis2011/183>.
- Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *The Journal of Systems and Software*, 85(6), 1222-1238. doi: 10.1016/j.jss.2012.02.017
- Strode, D. E., & Huff, S. L. (2012). A taxonomy of dependencies in agile software development. *Proceedings of the 23rd Australasian Conference on Information Systems, ACIS 2012, December 2012, Geelong, Australia* (forthcoming)
- Suddaby, R. (2010). Editor's comments: Construct clarity in theories of management and organisation. *Academy of Management Review*, 35(3), 346-357.
- Sutton, R. I., & Staw, B. M. (1995). What theory is not. *Administrative Science Quarterly*, 40(3), 371-384.
- Swanson, E. B. (1994). Information systems innovation among organizations. *Management Science*, 40(9), 1069-1088.
- Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard Business Review*, 64(1), 137-146.
- Thomas, D. R. (2006). A general inductive approach for analyzing qualitative evaluation data. *American Journal of Evaluation*, 27(2), 237-246. doi: 10.1177/1098214005283748
- Thomas, G., & Fernandez, W. (2008). Success in IT projects: A matter of definition. *International Journal of Project Management*, 26(7), 733-742.
- Thompson, J. D. (1967). *Organization in Action*. Chicago: McGraw-Hill.
- Toffolon, C., & Dakhli, S. (2000). A framework for studying the coordination process in software engineering *Proceedings of the 2000 ACM Symposium on Applied Computing, Como, Italy*. New York, NY: ACM.
- Truex, D. P., Baskerville, R., & Klein, H. K. (1999). Growing systems in emergent organisations. *Communications of the ACM*, 42(8), 117-123.
- Truex, D. P., Baskerville, R., & Travis, J. (2000). Amethodical systems development: The deferred meaning of systems development methods. *Accounting, Management and Information Technologies*, 10(1), 53-79.
- Tuomi, I. (1999). *Corporate Knowledge: Theory and Practice of Intelligent Organizations*. Helsinki: Metaxis.
- Turk, D., France, R., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management*, 16(4), 62-87.
- Van de Ven, A. H., Delbecq, A. L., & Koenig, R. (1976). Determinants of coordination modes within organizations. *American Sociological Review*, 41(2), 322-338.

- van Slooten, K. (1996). Situated method engineering. *Information Resources Management Journal*, 9(3), 24-31.
- Wagstrom, P., & Herbsleb, J. D. (2006). Dependency forecasting in the distributed agile organization. *Communications of the ACM*, 49(10), 55-56. doi: 10.1145/1164394.1164420
- Walsham. (2006). Doing interpretive research. *European Journal of Information Systems*, 15, 230-330. doi: 10.1057/ejis.1995.9
- Wang, W. P., Kleinman, D. L., & Luh, P. B. (2001). Modeling team coordination and decisions in a distributed dynamic environment. In G. M. Olson, T. W. Malone & J. B. Smith (Eds.), *Coordination Theory and Collaboration Technology* (pp. 673-710). Mahwah, New Jersey: Lawrence Erlbaum Associates.
- Wang, X., & Vidgen, R. (2007). *Order and chaos in software development: a comparison of two software development teams in a major IT company*. Paper presented at the Fifteenth European Conference on Information Systems, St Gallen, Switzerland.
- Wastell. (1996). The fetish of technique: Methodology as a social defence. *Information Systems Journal*, 6(1), 25-40.
- Weber, R. (2012). Evaluating and developing theories in the information systems discipline. *Journal of the Association for Information Systems*, 13(1), 1-30.
- Weick, K. E., & Roberts, K. H. (1993). Collective mind in organizations: Heedful interrelating on flight decks. *Administrative Science Quarterly*, 38(3), 357-381.
- West, D., & Grant, T. (2010). Agile development: Mainstream adoption has changed agility. 1-20. Retrieved from http://www.osp.ru/netcat_files/18/10/h_d8eddd303b6cf0c38c23601c4363bee4
- Westrup, C. (1993). Information systems methodologies in use. *Journal of Information Technology*, 8, 267-275.
- Whetten, D. A. (1989). What constitutes a theoretical contribution? *The Academy of Management Review*, 14(4), 490-495.
- Whittaker, S., & Schwarz, H. (1999). Meetings of the board: The impact of scheduling medium on long term group coordination in software development. *Computer Supported Cooperative Work*, 8(3), 175-205. doi: 10.1023/A:1008603001894
- Williams, L. (2010). Agile software development methodologies and practices. In M. V. Zelkowitz (Ed.), *Advances in Computers* (Vol. 80, pp. 1-44). Amsterdam: Elsevier.
- Woit, D. M. (2005). Requirements interaction management in an extreme programming environment: A case study. *Proceedings of the 27th International Conference on Software Engineering, ICSE 2005* (pp. 489-494). New York: ACM Press.
- Wynekoop, J. L., & Russo, N. L. (1995). Systems development methodologies: Unanswered questions. *Journal of Information Technology*, 10(2), 65-73.
- Wynekoop, J. L., & Russo, N. L. (1997). Studying system development methodologies: An examination of research methods. *Information Systems Journal*, 7(1), 47-65.

- Xia, W., & Lee, G. (2005). Complexity of Information systems development projects: conceptualization and measurement development. *Journal of Management Information Systems*, 22(1), 45-83.
- Xu, P., & Ramesh, B. (2007). Software process tailoring: an empirical investigation. *Journal of Management Information Systems*, 24(2), 293-328.
- Yang, D.-H., Kang, H.-R., & Mason, R. M. (2008). An exploratory study on the meta skills in software development teams: Antecedent cooperation skills and personality for shared mental models. *European Journal of Information Systems*, 17, 47-61.
- Yin, R. K. (1984). *Case Study Research* (1 ed.). Beverly Hills, CA: Sage.
- Yin, R. K. (2003a). *Applications of case study research* (2 ed. Vol. 34). Thousand Oaks: Sage Publications.
- Yin, R. K. (2003b). *Case Study Research* (3 ed.). Thousand Oaks: Sage.
- Yourdon, E. (1989). *Modern Structured Analysis*. Englewood Cliffs, New Jersey: Prentice Hall.
- Yuan, M., Zhang, X., Chen, Z., & Vogel, D. R. (2009). Antecedents of coordination effectiveness of software developer dyads from interacting teams: An empirical investigation. *IEEE Transactions on Engineering Management* 56(3), 494-507.

8. APPENDICES

8.1 APPENDIX A ETHICS APPROVAL

This appendix contains the ethics approval letter from Victoria University of Wellington, School of Information Management Human Ethics Approval Committee.

SIM HUMAN ETHICS COMMITTEE **Comments on Application for Human Ethics Approval**

Date: **19th October 2009**
Re: **Coordination in agile development projects**
Principal Researcher: **Diane Strode**
Supervisor (student research): **Professor Sid Huff (Primary)**
Associate Professor Sebastian Link (Secondary)
Ref No: **#16976**

Your HEC application has been reviewed and the Committee's decision is the following:

Approval Given after Amendments made – Application accepted.

Human Ethics Approval valid until: (Date: *as in application or no more than 3 years*)

Thank you for the amendments you have made to your HEC application. These meet the committee's required changes. On behalf of the HEC Chair I am authorised to inform you that you may now proceed with your research. You may begin your data collection immediately but please note that a hard copy of your application signed by both you and your supervisor (or other researchers involved for staff applications) is required within one month to ensure that we have a complete record of the approval of your application. This should be submitted to me at:

School of Information Management
Victoria University of Wellington
Kelburn Campus
Wellington

Wendy Chen
HEC Administrator
SIM Human Ethics Committee

SCHOOL OF INFORMATION MANAGEMENT
FACULTY OF COMMERCE AND ADMINISTRATION

8.2 APPENDIX B PARTICIPANT INFORMATION

This appendix contains the participant information sheet given to each participant before data collection began. The School of Information Management Research Ethics Committee approved this information sheet.

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

SCHOOL OF INFORMATION MANAGEMENT

Coordination in Agile Software Development Projects

Participant Information

My name is Diane Strobe and I am a PhD candidate in the School of Information Management at Victoria University of Wellington. This study will contribute to my PhD in Information Systems.

Purpose of the research

The purpose of this research is to investigate how agile software development practices contribute to the coordination of software development projects. Well-coordinated projects are important to successful software development and the contribution of agile software development practices to project coordination is not well understood.

Usefulness of the research

This research will provide information for those using agile methods and those considering adoption of these methods. Frequently in development projects decisions about which method and practices to adopt are based on anecdotal evidence or long-term experience. This research is designed to provide guidelines supported by sound research about which agile practices contribute to effective and flexible project coordination. In addition, the way the analysis will be carried out will provide information on which agile practices should be used together, those that can be used independently, those that are redundant in particular situations, and those that can be substituted successfully with equivalent alternative practices.

Why was your organisation selected for this study?

This study will involve a number of projects within different organisations, with different sizes of project, developing different types of system. Some projects will be using agile methods; some will be using a few practices from the agile methods, and in some projects agile methods will not be in use. Your organisation or project was selected because it fits one of these profiles.

The names of contact people were gathered from a number of sources including Internet search and personal contacts made at Wellington user groups and professional organisations. In some cases, the name was located in a newspaper or a computing magazine.

What participation will mean for you and your organisation

This research involves the study of individual software projects. The data for the study will be gathered only with your permission and that of your organisation. No discomfort or risk to you or your organisation is anticipated in this research.

The main way that data will be gathered is by interviewing people involved in a nominated software project. Interviews will take about one hour each and will be carried out at a time and place convenient to you and your organisation.

With permission, additional data will be sourced from:

- A questionnaire you will be asked to complete that gathers information about the project profile and practices used on the project.
- Observations of the workplace layout and the way that work activities are carried out.
- Publicly available artefacts including public web sites and annual reports.
- Examples of forms or documents used in the project.

All such data will remain confidential to the researcher and the PhD supervisors (identified below) and will be disposed of as detailed below.

What will you and your organisation gain from this research?

You and your organisation will be provided with:

- A descriptive summary of the project and its coordination aspects.
- A technical report summarising the whole study, the findings, and conclusions.
- Access to any published materials such as journal articles, conference papers, or practitioner reports generated from the data gathered in the study. In some cases these documents will be pre-publication versions.
- Access to the PhD thesis based on this research study.

Ethical considerations

This study has received ethical approval from the Victoria University of Wellington, School of Information Management Human Ethics Committee.

Confidentiality of the data

1. You and your organisation will not be identified in any published material based on this study.
2. The raw data will not be made available to your organisation or any other individual or organisation (except the researcher and the PhD supervisors as detailed below).
3. The raw data will be available to the researcher and the PhD supervisors and any transcriber employed to translate recorded interviews into text. Transcribers will be asked to sign a confidentiality agreement.
4. All printed data will be stored securely in locked filing cabinets. All data in electronic form will be stored in password-protected files. Data will be stored for two years after the final date of thesis submission (estimated to be the end of 2011). After that date, the written and electronic material will be destroyed.
5. The data provided will be used only for this research project and any further use will require your written consent.

Withdrawal from the study

You are not obliged to participate in any interview or provide any data. If you agree to participate you have the right to:

- Decline to answer any particular question.
- Ask for any audio recording to be turned off at any time during an interview.
- Withdraw from the study at any time and have the data you have contributed removed from the study and deleted, or returned to you. However, this will only be possible within six months of signing the Individual Participant Consent form.

Where will the results of this research be published?

This research data will be aggregated and will contribute to a publicly available PhD thesis. The findings may be published internationally in conference papers and journal articles targeting both practitioner and academic audiences. This means that your data will contribute to a better understanding of software development practices for software developers and project managers throughout the world.

Contact details of researcher and PhD supervisors

If you have any queries about the study please feel free to contact any one of us.

Researcher	PhD Supervisor	PhD Supervisor
Diane Strobe 04 463 5504 diane.strobe@vuw.ac.nz	Professor Sid Huff 04 463-5819 sid.huff@vuw.ac.nz	Associate Professor Sebastian Link 04 463-6813 sebastian.link@vuw.ac.nz
School of Information Management PO Box 600 Kelburn Wellington		

8.3 APPENDIX C CONSENT FORMS

This appendix contains two consent forms. The first form was completed by an appropriate person within an organisation and gave permission for project team members to be approached for data collection. Each participant in the nominated project completed the second form before data collection began.

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

SCHOOL OF INFORMATION MANAGEMENT

Coordination in Agile Software Development Projects

Consent Form - Organisation

I have read the Participant Information sheet and I have had the details of the study explained to me. My questions have been answered to my satisfaction and I understand that I can ask further questions at any time.

For the person who has authority to make such a decision

☐ I agree to allow the researcher to carry out their research in the following organisation.

Name of Organisation _____

☐ I agree that all people involved in the project nominated for this study can choose to participate without coercion.

Name _____

Signature _____

Role _____

Date _____

Page 1 of 1

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

SCHOOL OF INFORMATION MANAGEMENT

Coordination in Agile Software Development Projects

Consent Form for Individual Participant

I have read the Participant Information sheet and I have had the details of the study explained to me. My questions have been answered to my satisfaction and I understand that I can ask further questions at any time.

I understand that any information I provide will be kept confidential to the researcher, the supervisors, and the person who transcribes the tape recordings of our interview(s). The PhD thesis or in any other publications or conference presentations arising from this study, will not use my name, and no information that I provide will be attributed to me, any other individuals, the project that I am discussing, or the organisation itself. I understand that all printed data will be stored securely in locked filing cabinets and all data in electronic form will be stored in password-protected files.

I understand that the information I have provided will be used only for this research project and that any further use will require my written consent.

I understand that I can withdraw from this study within six months from the date this consent form is signed. If I withdraw, the data I have contributed will be removed from the study and deleted, or returned to me.

This consent form and the data gathered in the study will be held for a period of two (2) years after the thesis submission date.

I agree

- ☐ To participate in this study under the conditions set out in the Participant Information sheet.
- ☐ To be interviewed.
- ☐ To have my interview audio recorded and transcribed.

At the end of the two year period, I want to (select one)

- ☐ Have my audio files deleted by the researcher.
- ☐ Have my audio files returned to me.

Research findings

- ☐ I want to receive a summary of the findings from my project.
- ☐ I want to receive a summary of the findings from the whole research study.

Name _____

Signature _____

Date _____

8.4 APPENDIX D INTERVIEW PROTOCOLS

This appendix contains the interview protocols used during the interviews. This is the final form of the protocol after it was adjusted once the pilot interview was completed. There are two protocols; the first is for interviewing the project leader to gather project background information and information about coordination. The second is for interviewing other project team members and involved questions about their background and daily work practices.

V10
Thursday, 22 March 2012

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

Coordination in Agile Software Development Projects
Interview Outline
Project Manager

Protocol for interview with project manager/team leader

1. Collect the questionnaire on project profile and software development practices
2. Background questions
 - Name
 - Job title
 - Years of IT experience
 - Educational background
3. Organisation background - include
 - Employees
 - IT employees
 - SD employees
 - Organisation structure (organisation chart if available)
4. Role in project – focus on THIS PROJECT
5. Briefly describe the project
 - Purpose
 - History
 - How it is structured (team members and their roles, staff turnover)
 - Importance to the organization
 - Completion status
 - Length of time working together
 - How did you and the team learn about agile methods
 - How/why did you tailor of method?
 - Architecture
 - Project criticality
 - Project success assessment
6. Project profile and process
 - Review questionnaire of project profile details and agile practices here
 - Overall project process (initiation to conclusion) – begin at the end and work forwards
 - Consider the iteration length and work activities
 - Consider the daily work activities
 - Any other activities?
 - Who is involved (stakeholders)
 - How does your team communicate with its 'customer'
 - What are the main resources (products/partial products/artifacts used)
 - How is work allocated among team members?

- What technologies are used to coordinate (communicate) work activities?
 - What forms or documented procedures are used?
 - What forms or documented procedures are created?
7. Coordination
 - What do you think makes THIS project a well-coordinated project?
 - Can you provide an example from this project?
 - What interferes with the coordination on THIS project?
 - Can you provide examples of problems encountered to date?
 - Type of problem, source of problem, reasons for occurrence
 - Based on all of your past software project experience, how would you define a 'well-coordinated' project?
 8. Which practices/activities, agile or not, do you think are particularly useful and what makes them useful?
 - Aim for explicit examples/evidence
 9. Anything else to add?
 10. Discuss appropriate interviewees and next steps

Interview Outline

Team Member

1. Protocol for individual interview of team member
2. Collect the questionnaire on agile practices from at least ONE team member

I take a very broad view of coordination. I treat coordination as *anything* you do to contribute to a project that is not directly writing code. So even if you are commenting the code, this could be considered a type of coordination, because you are communicating (which is one type of coordination) with future developers who will need to read the code.

3. Background questions
 - Name
 - Job title
 - Years of IT experience
 - Educational background
 - Describe your job (what is your role) and its goals
4. What are your main work activities (3-5) IN THIS PROJECT
 - For each activity (this will depend on specialisation/generalisation of work)
 - Main purpose of these activities – each one?
 - Depending if work is broken down into distinct activities or not...
 - How is work assigned to you?
 - How do you know when to start the work/activity?
 - Who do you work with to complete the activity [stakeholder identification]
 - Who do you send communications to?
 - Who do you receive communications from?
 - How do you share out/or delegate the work
 - How do you decide who to share out or delegate the work to
 - What resources (things or information) do you need to complete the activity
 - What technologies do you use to help you carry out the activity? (email, config man. tools, wiki, project database, on-line project plan, on-line specifications...)
 - What forms or documents do you need to perform the activity? Examples?
 - What is the product or partial product of the activity? (documents, information, software)
 - Who relies on the product of this activity?
 - Do any of your work products need to be integrated or fit in with other peoples work or applications?
 - Who waits for your work to be completed? Example?
 - Individual activities:
 - How do you know when the activity is complete?
 - How do others know when the activity is complete?
 - What things hinder this activity
 - What do you wait for?
 - Negotiate for?
 - Bid for?
 - What would happen if this activity was not carried out?
 - What alternative ways could the outputs or goal of this activity be achieved?
5. Dependency prompts:
 - Vendor [stakeholder identification]
 - Customer [stakeholder identification]

- * Other business units [stakeholder identification]
 - * Lifecycle (only when multiple teams are working on same product for different platforms) – how is this managed/organised?
 - * Big picture - how do you achieve an overview of how all of the parts of the system fit together?
 - * Testing – test versions - how do you manage/organise this? What types of testing do you do?
 - * Parallel development – when two or more people work on the same code module – how do you manage/organise this?
 - * Change – when changes are made how does this impact other code modules, documentation, testing
 - * Expertise – shared product knowledge
 - How do you all know what the product must do?
 - How do you know what the other people on the team are doing on a daily basis within the code?
 - How do you know what the other people on the team are doing on a daily basis within the documentation?
 - How do you know what the other people on the team are doing on a daily basis within the test bank?
 - How do you come to know what the skills and capabilities of the other team members are – how do you know who to ask about things?
 - * Historical understanding – how do you find out about previous decisions made that impact
 - The code
 - This project?
 - * Integration – how do you integrate the code and other components
 - regularly, randomly, at the end? To a schedule?
 - * What is the main source of bugs in the system?
6. Coordination
- What do you think makes THIS project a well-coordinated project?
 - * Can you provide an example from this project?
 - What interferes with the coordination on THIS project?
 - * Can you provide examples of problems encountered to date?
 - * Type of problem, source of problem, reasons for occurrence
 - Based on all of your past software project experience, how would you define a 'well-coordinated' project?
7. Which practices/activities, agile or not, do you think are particularly useful and what makes them useful?
- Aim for explicit examples/evidence
8. Anything else to add?
9. Discuss appropriate interviewees and next steps.

8.5 APPENDIX E THE QUESTIONNAIRES

This appendix provides copies of the two questionnaires, the Project Profile and Software Development Practices Questionnaire, and The Software Development Practices Questionnaire.

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

Coordination in Agile Software Development Projects

Questionnaire

Project Profile and Software Development Practices Questionnaire

This questionnaire gathers basic information about a single software development project, the organisation carrying out the project, and the practices used in that project.

This questionnaire is for a person in the organisation who works with a team of IT developers as a **supervisor, team leader** or **project manager** (or has a similar job role).

The questionnaire should take about 20 minutes to complete.

You are not obliged to complete this questionnaire.

You cannot be identified in this questionnaire and the information gathered will not be made available to your organisation or any other individual or organisation.

Further information about this study is provided in the **participant information sheet** provided with this questionnaire.

Section 1 General organisation background

This section gathers information about your organisation. Your organisation means your department, section, organisational unit or major work group.

1. What type of development does your organisation undertake?

Tick all that apply

- ☐ We provide specific products for external customers by:
 - ☐ completing one-off contracts
 - ☐ mass production of shrink-wrapped products
 - ☐ customising pre-developed software
 - ☐ providing service/support of software systems
- ☐ We provide in-house development to support the running of the organisation by:
 - ☐ carrying out our own development
 - ☐ customisation of brought-in products
- ☐ We develop software for inclusion in our organisation's products by:
 - ☐ in-house development
 - ☐ purchasing software from an outside source
 - ☐ other, please specify _____

2. Approximately how many staff are involved in software development in this organisation?

_____ OR if you don't know please tick this box ☐

3. Approximately how many projects has your organisation completed using an agile method?

_____ OR if you don't know please tick this box ☐

OR we are not using agile methods ☐

Section 2 Project description

The rest of this questionnaire gathers information about a single project. Please select one typical project which was recently completed or is currently underway within your organisation which is using an agile method. If you are not using an agile method choose a project that you are currently working on or have recently completed.

10. What is the name of the project? _____

11. If your project is completed: what was the duration of the project? _____

Otherwise: what is the estimated duration of the project? _____

12. What is the nature of this project?

Select **one or more**

- ☐ new system
- ☐ enhancement to existing system
- ☐ replacement of existing system
- ☐ maintenance
- ☐ other, please specify _____

13. What criteria do you use to determine when this project is completed?

Select **one only**

- ☐ system is handed over to testers
- ☐ client sign off
- ☐ system moves into maintenance phase
- ☐ system is in use with its intended user base
- ☐ other, please specify _____

14. How many **full-time** developers (programmers, analysts, testers etc.) are involved in this project?

15. How many **part-time** developers (programmers, analysts, testers etc.) are involved in this project?

16. Approximately how many end-users will use the system developed in this project?

_____ OR if you don't know please tick this box ☐

17. Briefly describe the purpose of the project

Examples To enable restaurant bookings to be captured on-line
To provide on-line capabilities for staff to diarise future events

18. What business area does the project application address?

Select **one or more**

- ☐ accounting
- ☐ banking
- ☐ entertainment (other than games)
- ☐ gaming
- ☐ inventory
- ☐ marketing
- ☐ management information
- ☐ office automation
- ☐ purchasing
- ☐ sales
- ☐ systems software
- ☐ scientific
- ☐ engineering
- ☐ other, please specify _____

19. What type of system is under development in this project?

Select **one or more**

- ☐ decision support system
- ☐ management information system
- ☐ office automation system
- ☐ transaction processing
- ☐ systems software
- ☐ intranet system
- ☐ Internet system
- ☐ operating system
- ☐ extranet system
- ☐ other, please specify _____

20. Which of the following are currently used in the final application created in this project

Select **one or more**

- ☐ gaming software engines
- ☐ interface (console)
- ☐ interface (web)
- ☐ interface (windows)
- ☐ mainframe development
- ☐ middleware
- ☐ mobile devices
- ☐ object database

- ☐ object-oriented concepts
- ☐ object-oriented languages
- ☐ relational database
- ☐ web development (dynamic)
- ☐ web development (static)
- ☐ other, please specify _____

21. Please name the tools used in this project

If you use more than one tool in each category just name the main two tools used

Case/modelling tools	_____
Unit testing tools	_____
System testing tools	_____
Programming languages	_____
Other tools	_____

22. How long has this organisation been using this particular agile method or technique?

For example: Extreme programming, Scrum, Agile methods, prototyping

- ☐ We don't use an agile method
- ☐ Never before
- ☐ Up to 6 months
- ☐ Up to 1 year
- ☐ Up to ____ years (*state how many years*)

23. How many projects has this team previously worked on together?

- ☐ Never before
- ☐ One project
- ☐ Two to five projects
- ☐ More than five projects

Section 3 Agile method usage

24. Which of the following techniques do you use in this project?

Please circle the number to show that you use a technique and how frequently you use it.

If you don't know what the technique is, or don't know if your team uses the technique, then tick the 'don't know' column.

Technique	Extensively	Often	Seldom	Never	Don't know
Concurrent development (analysis, design, code, and test carried out simultaneously)	3	2	1	0	
Iterative development	3	2	1	0	
Time boxing (iterations of set length)	3	2	1	0	
Incremental development	3	2	1	0	
Evolutionary prototyping	3	2	1	0	
Small releases of software product	3	2	1	0	
Test first development	3	2	1	0	
Daily builds of complete system	3	2	1	0	
Automated regression testing	3	2	1	0	
Refactoring of code	3	2	1	0	
Testing throughout each iteration (including unit, integration and acceptance testing)	3	2	1	0	
Customer on-site	3	2	1	0	
Method coach on site	3	2	1	0	
Tester(s) collocated with team	3	2	1	0	
Rooms organised for pair programming	3	2	1	0	
Whole team works in same office or floor	3	2	1	0	
Pair programming	3	2	1	0	
Coding to an agreed standard	3	2	1	0	
Collective ownership of code	3	2	1	0	
40 hour week	3	2	1	0	
Sprint Goal	3	2	1	0	
Daily team meetings	3	2	1	0	
Iteration planning meeting	3	2	1	0	
Planning game (meeting with stakeholders at start of each iteration to plan scope)	3	2	1	0	
Reflective workshops for methodology adaptation at each iteration	3	2	1	0	
User stories	3	2	1	0	
System metaphor developed	3	2	1	0	
Only what has direct business value is	3	2	1	0	

Technique	Extensively	Often	Seldom	Never	Don't know
developed					
Requirements are prioritised with the customer or user	3	2	1	0	
Changes to requirements are negotiated with the users to maintain time frames	3	2	1	0	
Joint Application Development (JAD) sessions (requirements gathering sessions with selected stakeholders)	3	2	1	0	
Design is kept as simple as possible	3	2	1	0	
Coded solution is kept as simple as possible	3	2	1	0	
Product Backlog	3	2	1	0	
Sprint Backlog	3	2	1	0	
Release Backlog	3	2	1	0	
Milestones to track progress	3	2	1	0	
Product Backlog Graph metric	3	2	1	0	
Sprint Backlog Graph metric	3	2	1	0	
Please add any other software development techniques you use on this project -					
1.	3	2	1		
2.	3	2	1		
3.	3	2	1		
4.	3	2	1		
5.	3	2	1		

Please circle yes or no for these questions:

25. We will carry out a project post mortem yes / no
26. A feasibility study has been carried out yes / no
27. A business study has been carried out yes / no
28. A resource requirements analysis has been carried out yes / no

29. What is the typical length of iterations in this project?

- ☐ No iterations
- ☐ 1 week
- ☐ 2 week
- ☐ 1 month
- ☐ Flexible length
- ☐ Other, please specify _____

30. Which agile method did you use in this project?

You may select more than one

- ☐ We do not use an agile method
- ☐ Generic agile method
- ☐ Extreme Programming (XP)
- ☐ Scrum
- ☐ We created a unique methodology at the start of the project
- ☐ We tailored an existing methodology at the start of the project
- ☐ Other, please specify _____

Thank you for taking part in this study

The researcher will collect this questionnaire from you in person

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

Coordination in Agile Software Development Projects

Questionnaire

Software Development Practices Questionnaire

This questionnaire gathers basic information about a single software development project and the practices used in that project.

This questionnaire is for a person in the organisation who works within a team of IT developers as a **developer, business analyst, test analyst** (or has a similar job role).

The questionnaire should take about 10 minutes to complete.

You are not obliged to complete this questionnaire.

You cannot be identified in this questionnaire and the information gathered will not be made available to your organisation or any other individual or organisation.

Further information about this study is provided in the **participant information sheet** provided with this questionnaire.

Agile method usage

1. Which of the following practices and techniques do you use in this project?

Please circle the number to show that you use a technique and how frequently you use it.

If you don't know what the technique is, or don't know if your team uses the technique, then tick the 'don't know' column.

Technique	Extensively	Often	Seldom	Never	Don't know
Concurrent development (analysis, design, code, and test carried out simultaneously)	3	2	1	0	
Iterative development	3	2	1	0	
Time boxing (iterations of set length)	3	2	1	0	
Incremental development	3	2	1	0	
Evolutionary prototyping	3	2	1	0	
Small releases of software product	3	2	1	0	
Test first development	3	2	1	0	
Daily builds of complete system	3	2	1	0	
Automated regression testing	3	2	1	0	
Refactoring of code	3	2	1	0	
Testing throughout each iteration (including unit, integration and acceptance testing)	3	2	1	0	
Customer on-site	3	2	1	0	
Method coach on site	3	2	1	0	
Tester(s) collocated with team	3	2	1	0	
Rooms organised for pair programming	3	2	1	0	
Whole team works in same office or floor	3	2	1	0	
Pair programming	3	2	1	0	
Coding to an agreed standard	3	2	1	0	
Collective ownership of code	3	2	1	0	
40 hour week	3	2	1	0	
Sprint Goal	3	2	1	0	
Daily team meetings	3	2	1	0	
Iteration planning meeting	3	2	1	0	
Planning game (meeting with stakeholders at start of each iteration to plan scope)	3	2	1	0	
Reflective workshops for methodology adaptation at each iteration	3	2	1	0	
User stories	3	2	1	0	
System metaphor developed	3	2	1	0	
Only what has direct business value is	3	2	1	0	

Technique	Extensively	Often	Seldom	Never	Don't know
developed					
Requirements are prioritised with the customer or user	3	2	1	0	
Changes to requirements are negotiated with the users to maintain time frames	3	2	1	0	
Joint Application Development (JAD) sessions (requirements gathering sessions with selected stakeholders)	3	2	1	0	
Design is kept as simple as possible	3	2	1	0	
Coded solution is kept as simple as possible	3	2	1	0	
Product Backlog	3	2	1	0	
Sprint Backlog	3	2	1	0	
Release Backlog	3	2	1	0	
Milestones to track progress	3	2	1	0	
Product Backlog Graph metric	3	2	1	0	
Sprint Backlog Graph metric	3	2	1	0	
Please add any other software development techniques you use on this project -					
1.	3	2	1		
2.	3	2	1		
3.	3	2	1		
4.	3	2	1		
5.	3	2	1		

Please circle yes or no for these questions:

2. We will carry out a project post mortem	yes / no
3. A feasibility study has been carried out	yes / no
4. A business study has been carried out	yes / no
5. A resource requirements analysis has been carried out	yes / no

6. What is the typical length of iterations in this project?

- ☐ No iterations
- ☐ 1 week
- ☐ 2 week
- ☐ 1 month
- ☐ Flexible length
- ☐ Other, please specify _____

7. Which agile method did you use in this project?

You may select more than one

- ☐ We do not use an agile method
- ☐ Generic agile method
- ☐ Extreme Programming (XP)
- ☐ Scrum
- ☐ We created a unique methodology at the start of the project
- ☐ We tailored an existing methodology at the start of the project
- ☐ Other, please specify _____

Thank you for taking part in this study

The researcher will collect this questionnaire from you in person

8.6 APPENDIX F DATA SOURCES

This appendix lists all data collection points for each case in Table 56. A kick off meeting was the initial meeting where project selection was negotiated with someone in the organisation. Only those projects used in the research are shown. When people were interviewed twice this means that the interview was carried out in two separate sessions.

Key for Table 56

DS - Transcribed by researcher

TS - Transcribed by professional transcription service

FN - Field notes

** - not transcribed (pilot case interview)

Interview data

Total number of formal interviews - 15

Total number of people interviewed - 16 (one double interview)

Table 56 Data sources

	Project Code Project name Development methodology	Kickoff session	Interview dates Participant code Role	Interview length	Transcribed by	Comment
1	A Hour (Pilot Study) 20 Hours SDLC	By email	25 Nov 09 IT Manager AP01	40min	DS	Pilot study Interviewed on-site in participants office
2			27 Dec 09 AT01 Web Developer	40min	**	Interviewed on-site in meeting room
3			4 Dec 09 AT02 Oracle Developer	42min	DS	Interviewed at participants home
4	B Land Concessions online Agile - scrum	7 Dec 09 Senior Systems Analyst & Project Manager	14 Dec 09 BP01	1:30min	DS	Interviewed on-site in meeting room
5			9 Mar & 24 Mar 2010 BT01 Senior Developer	1:10min	DS	Interviewed on-site in meeting room Interviewed in coffee shop
6	C Storm Migration project Agile – Scrum/XP	8 Dec 09 Programme Manager	15 Dec 09 CP01 Programme Manager	1:30min	TS	Interviews and data gathering are now complete for this project
7			11 & 17 Mar 2010 CT01 Developer	1:43min	DS	Interviewed at University meeting room
8			18 Mar 2010 CT02 Technical domain expert	1:10min	TS TS	Interviewed on-site in meeting room
9			18 Mar 2010 CT03 Developer	45min	DS	Interviewed at University meeting room
10			22 Mar 2010 CT04 Tester	1:22min	TS	Interviewed at University meeting room
11	D Silver Registration system Agile - Scrum	11 Dec 09 Managing Director, Scrum coach, Senior Developer	11 Dec 09 Double interview DP01	1:38mn	DS	Interviewed on-site in meeting room
12			17 Feb 2010 Developer DT01 *5	1:10min	TS	Interviewed on-site in meeting room
13			24 Feb 2010	20min	TS	Interviewed on-site in

	Project Code Project name Development methodology	Kickoff session	Interview dates Participant code Role	Interview length	Transcr -ibed by	Comment
			Senior Developer DT02			meeting room
14			24 Feb 2010 25 Feb 2010 Developer DT03	1:40min	TS	Interviewed on-site in meeting room
15	G Rock Online statements Hybrid	By email	18 Apr 2011 GP01 Senior Business Analyst	1:10hr	TS	Interviewed on-site in meeting room
16			26 Apr 2011 GT01 Senior Analyst Programmer	1:15 hr	TS	Interviewed on-site in meeting room
17			27 Apr 2011 GT02 Test Analyst – external contractor	1:38hr	TS	Interviewed on-site in meeting room
18			4 May 2011 GT03 Technical Designer	1 hr	TS	Interviewed on-site in meeting room

8.7 APPENDIX G GLOSSARY OF TERMS

This appendix is a glossary of terms identified in the case analyses and used in the theory of coordination presented in this thesis.

Key theoretical concepts

Coordination - 1) the harmonious organisation of activities carried out to achieve common goals, more specifically 2) the managing of dependencies in a situation.

Dependency - is when the progress of one action relies upon the timely output of a previous action, or the presence of some specific thing. Dependencies lead to potential or actual constraints on projects. Potential constraints are those that are currently organised or managed well, causing no problems in the progression of a project. Actual constraints are bottlenecks or points in a project that stakeholders are aware of, but have no immediate means to circumvent. This is also called an unmanaged dependency.

Coordination mechanism - an entity (person or artifact) or activity (practice or technique) addressing one or more dependencies in a situation

Coordination strategy - a group of coordination mechanisms used in a situation. Mechanisms form a strategy when they are selected consciously by project stakeholders, rather than occurring by chance

Coordination effectiveness - the extent to which a set of coordination mechanisms address the dependencies in a project. This is the initial definition given in the conceptual framework section. Clarifying this definition was one aim of the study. The full definition is therefore provided in Chapter 5 A Theory of Coordination in Agile Projects

Antecedents

Project complexity - a project property whereby the project consists of many varied interrelated parts (Baccarini, 1996)

Project uncertainty - a project quality which is the “perceived level of not knowing the appropriate course of action and/or its outcome at a given point in time” (Madsen, 2007, p. 858).

Organisation structure - the way the functional units of an organization are compartmentalized to meet organizational goals (Galbraith, 1974)

Coordination strategy components

Synchronisation - the relation that exists when things occur at the same time. A dictionary definition for synchronise is “occur at the same time; be simultaneous with” (Allen, 1990, p. 1236)

Synchronisation activity - an activity involving all team members that brings them together at the same time and place for some pre-arranged purpose

Synchronisation artefact - a physical thing generated during synchronisation activities that contains information used by all team members in accomplishing their work. These artefacts store project information such as design decisions and may serve as objects to focus discussion. The nature of an artefact can be publicly visible to the whole team at a glance (e.g. on a whiteboard in the team workroom) or largely invisible but available (e.g. located in a shared computer file). An artefact can be physical or virtual, temporary or permanent

Structure - the physical arrangement of, and relations between, the people participating in the project

Proximity - the physical closeness of individual team members. A team can include development team members, customer team members, or other stakeholders. Proximity can range from adjacent desks, to members located in different rooms, floors of a building, different buildings, different countries, or time zones

Availability - team members are continually present and able to respond to requests for

assistance, information, and to participate in project activities when needed

Substitutability – a team member has the expertise and skills to perform the tasks of another to maintain the project schedule

Boundary spanning - the act of linking two or more groups of people separated by location, hierarchy, or function (Levina & Vaast, 2005)

Boundary spanning activity - an activity performed to enable reciprocal interaction between project team members and other external parties to the project to meet project goals (Levina & Vaast, 2005)

Boundary spanning artefact - an artefact produced to support boundary spanning 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 e.g. software prototype, engineering sketch (Levina & Vaast, 2005)

Coordinator role - the role taken by a person on a development team who initiates and sustains reciprocal interaction with other external parties to the project to meet project goals (Hoda, et al., 2010)

Coordination effectiveness components

Implicit coordination - occurs when team members anticipate the actions and needs of their colleagues and adjust their behaviour accordingly without preplanning or direct communication (Nonaka, 1994; Rico, et al., 2008; W. P. Wang, et al., 2001)

Explicit coordination - occurs when two or more team members send communication messages to one another using formal or informal, oral or written, transactions to integrate their work (Nonaka, 1994; Rico, et al., 2008; W. P. Wang, et al., 2001)

Dependency categories

Primary dependency - a dependency identified directly from case data that addresses one or more dependencies in a situation

Secondary dependency - a collective name for one or more primary dependencies

Managed dependency - a dependency for which there is an identifiable coordination mechanism

Unmanaged dependency - a dependency for which there is no identifiable coordination mechanism.

Requirement dependency - a situation wherein domain knowledge or a requirement is not known and must be located or identified and this affects progress. This is a form of knowledge dependency.

Expertise dependency - a situation wherein technical or task information is known only by a particular person and this affects progress. This is a form of knowledge dependency.

Historical dependency - a situation where in knowledge about past decisions is needed and this affects project progress. This is a form of knowledge dependency.

Task allocation - a situation wherein who is doing what, and when, is not known and this affects progress. This is a form of knowledge dependency.

Activity dependency - a situation where in an activity cannot proceed until another activity is complete and this affects project progress. This is a form of task dependency.

Business process dependency - a situation where in an existing business process causes activities to be carried out in a certain order and this affects progress. This is a form of task dependency.

Technical dependency – a situation where in a technical aspect of development affects progress, such as when one software component must interact with another software component and its presence or absence affects project progress.

Entity dependency - a situation where in a resource (person, place or thing) is not available and this affects progress. This is a form of resource dependency.

8.8 APPENDIX H SUMMARY OF XP AND SCRUM

This appendix summarises two of the most commonly adopted agile methods and their practices in Table 57 and Table 58. Both summaries are adapted from Strode (2005, Appendix K).

Table 57 Summary of Extreme programming

Extreme programming (XP)	
This analysis is based on Beck (2000)	
Extreme programming is a system development methodology focusing on practices for organising software development in a team environment. "The prime objectives of the method are; to develop the system as rapidly as possible; to produce software which meets the customer's needs and has some acceptable level of quality while mitigating risks; to produce quality software in small efficient teams using established software development techniques; to have high morale and a good working environment for developers. Software quality is important. It is negotiated with the customer and supported with specific techniques." (Strode, 2005, Appendix K: p. 15)	
First published	Beck (1999)
Major publication	Beck (2000) Beck and Andres (2005)
Country	USA
Paradigm	Objectivist with emergent properties
Assumptions and values	<p>Values are: communication, simple solutions, constant feedback, courage</p> <p>Project variables are: cost, time, quality, scope</p> <p>Assumptions</p> <ul style="list-style-type: none"> • Cost-of-change no longer valid • Change is constant during a project <p>Principles</p> <p>Rapid feedback, assume simplicity, incremental change, embracing change, quality work, teaching learning, small initial investment, play to win, concrete experiments, open, honest communication, work with peoples instincts, accepted responsibility, local adaptation, travel light, honest measurement</p>
Perspective	Programmer centred
Objectives	<p>Only technological solutions are considered</p> <p>Aim - To reduce development time, respond effectively to business and technology changes, to maintain and improve competitiveness, improve team productivity, address project risk, make software development fun, have a better relationship with customers, have a stable programmer team with high morale and a good working environment, produce software at some negotiated level of quality</p>
Domain	Problem-solving methodology for well-defined problems
Target	<ul style="list-style-type: none"> • New projects • Projects with vague and/or changing requirements • Small projects with 2-10 programmers • Object-oriented concepts and programming languages • Outsourced software development • Fix-price contracts • In-house development • Applications • Application frameworks • Software applications • Web-based systems

	<ul style="list-style-type: none"> • Shrink-wrapped software • Not well suited to development for reuse
Model	None, knowledge of object modelling techniques assumed
Techniques: practices and metrics	<ul style="list-style-type: none"> • Pair programming • Planning game • Simple design and refactoring • Small releases • Iterative and incremental development • 1-4 week iterations • Coding standards • Collective ownership and pair programming • Continuous integration and testing • 40 hour week • System metaphor • On-site customer • Metrics • Room arrangements • Listening • Designing • User stories
Tools	Automated unit testing and integration
Scope	<p>Development is iterative and incremental</p> <p>Phases</p> <ul style="list-style-type: none"> • Exploration • Planning • Iteration to first release • Productionizing • Maintenance • Death <p>All phases contain analysis, design, coding, testing</p>
Practice	<p>Background: Practitioner-based</p> <p>Roles:</p> <ul style="list-style-type: none"> • Programmer • Tracker • Coach • Consultant • Big Boss • Customer <p>Difficulties - big teams, distrustful customers, ungraceful change (non-object oriented systems), hierarchical business culture, achieving collaboration, achieving simplicity</p> <p>Skill levels - average programmers, on-site coach required</p>
Product	Working software, code and unit test suite delivered in monthly increments
Tailorability	Local adaptation acceptable

Table 58 Summary of Scrum

Scrum	
This analysis is based on (Schwaber & Beedle, 2002)	
Scrum is a system development methodology focusing on project management practices for organising software development in a team environment. Project management in Scrum “involves regular inspection of development activities to observe the state and progress of development, and then adjustment of the activities to produce the desired and predicted outcomes.” (Strode, 2005, Appendix K: p. 26)	
First published	Schwaber (1995)
Major publication	Schwaber and Beedle (2002)
Country	USA
Paradigm	Objectivist with emergent properties
Assumptions and values	<p>Empirical process control</p> <p>Assumptions</p> <ul style="list-style-type: none"> • Software creation is like new product development • The process is unstable • Teams are self-organising • Development processes are overlapping • ‘Multilearning’ is needed • Subtle process control is needed • Organisational transfer of learning occurs <p>Team qualities: focus, openness, respect, courage</p> <p>Project variables are: cost, time, quality, functionality</p> <p>Control is achieved by adjusting functionality</p>
Perspective	Project manager and programmer centred
Objectives	<p>Only technological solutions are considered</p> <p>Aims</p> <p>To provide techniques for project management, to support self-empowered teams, to give a global view of development, to produce complex, sophisticated software that meet s business needs, to provide productivity gains, to produce working functionality within one month and in monthly increments thereafter, to act as a wrapper for other software engineering practices, to deliver an increment of software by controlling functionality</p>
Domain	Problem-solving methodology for well-defined problems
Target	<p>Large scale programming (that can be broken down into smaller development teams</p> <p>Complex projects</p> <p>Projects with vague requirements</p> <p>Projects with constant requirements changes</p> <p>Projects that can be carried out by 5-9 programmers</p> <p>Applications</p> <p>Application frameworks designed for reuse</p> <p>Web deployed wireless technologies</p> <p>Web-based systems</p> <p>Object oriented systems</p> <p>General method suitable for any environment</p>
Model	None, knowledge of object modelling techniques assumed

Techniques: practices and metrics	<p>Techniques are designed to enable management to carry out control by observation and incremental adjustment leaving the team to carry out development unhindered</p> <p>Techniques</p> <ul style="list-style-type: none"> • Product Backlog • Sprint • Sprint Goal • Sprint Backlog • Sprint Planning meeting • Daily Scrum • Sprint Review meeting • Release Backlog • Customer on-site • Work space configuration • Daily builds and tests • Testing (all types) <p>Metrics: Product Backlog Graph, Sprint Backlog Graph</p>
Tools	None
Scope	<p>Development is iterative and incremental</p> <p>Phases</p> <ul style="list-style-type: none"> • Startup • Iterations of: Sprint planning meeting, Sprint, Sprint review meeting
Practice	<p>Background: Practitioner-based method</p> <p>Roles: Scrum master, Product Owner, Scrum team</p> <p>Difficulties: None</p> <p>Skill levels: Not stated</p>
Product	Working software delivered in monthly increments
Tailorability	<ul style="list-style-type: none"> • No advice on tailorability • Scalable to large projects • Can be used with XP or other engineering practices without tailoring

8.9 APPENDIX I LAND CASE DESCRIPTION

A project participant has checked this description to ensure its accuracy, and the anonymity of participants and organisations.

The case descriptions are available in a file named *Strode PhD Case Descriptions.pdf*.

This file contains appendices I, J, K, and L, and can be viewed and downloaded from the following location:

Location: <http://www.box.net>

Username: StrodePhD@gmail.com

Password: PhDResearch

8.10 APPENDIX J STORM CASE DESCRIPTION

A project participant has checked this description to ensure its accuracy, and the anonymity of participants and organisations. See Appendix I for the location of this file.

8.11 APPENDIX K SILVER CASE DESCRIPTION

A project participant has checked this description to ensure its accuracy, and the anonymity of participants and organisations. See Appendix I for the location of this file.

8.12 APPENDIX L ROCK CASE DESCRIPTION

A project participant has checked this description to ensure its accuracy, and the anonymity of participants and organisations. See Appendix I for the location of this file.